VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

AUTOMATICKÉ GENEROVÁNÍ PROJEKTU V PROSTŘEDÍ TIA PORTAL
AUTOMATIC PROJECT CODE GENERATION IN TIA PORTAL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Roman Halata**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Radek Štohl, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Roman Halata ***ID:*** 186074
Ročník: 2 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Automatické generování projektu v prostředí TIA portal

POKYNY PRO VYPRACOVÁNÍ:

1) Vypracujte přehled dostupných nástrojů pro automatické generování kódu PLC Siemens a zhodnoťte.
2) Vypracujte přehled a možnosti knihovny Siemens TIA Openness.
3) Na základě návrhu realizujte strukturu projektu v TIA (základní objekty FC/FB/DB/UDT) tak, aby umožňovala automatické generování.
4) Realizujte program pro automatické generování kódu s pomocí knihovny Siemens TIA Openness.
5) Ověřte funkčnost vlastního řešení.

DOPORUČENÁ LITERATURA:

SIEMENS. SIMATIC Automating projects with scripts: System Manual. Mnichov. SIEMENS, 2017.

Dle vlastního literárního průzkumu a doporučení vedoucího práce.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 1.6.2020

Vedoucí práce: Ing. Radek Štohl, Ph.D.
Konzultant: Ing. Lukáš Honc

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

ABSTRAKT

Diplomová práce se zabývá automatickým generováním kódu pro PLC od firmy Siemens. První část se zaměřuje na aktuálně dostupné nástroje pro generování kódu a možnosti C# knihovny Siemens TIA Openness. Dále je vytvořen návrh struktury projektu v TIA portal. Nakonec je navržena a vytvořena uživatelská aplikace pro automatické generování kódu v TIA portal v15, vhodná především pro větší projekty s rozdělením na jednotlivé stanice.

KLÍČOVÁ SLOVA

TIA portal, PLC, WPF, MVVM, Generování kódu, SQL, databáze, C#, Openness, automatizace, aplikace

ABSTRACT

The diploma thesis deals with an automatic code generation for PLC from Siemens. The first part focuses on currently available tools for code generation and options of C# library Siemens TIA Openness. Furthermore, a design for the project structure in the TIA portal is created. Finally, a user application for automatic code generation in TIA portal v15 is designed and created, especially suitable for larger projects that could be divided into individual stations.

KEYWORDS

TIA portal, PLC, WPF, MVVM, Code generation, SQL, database, C#, Openness, automation, application

HALATA, Roman. *Automatické generování projektu v prostředí TIA portal*. Brno, 2020, 74 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce: prof. Ing. Radek Štohl, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Automatické generování projektu v prostředí TIA portal“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: 1. června 2020 .....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Radkovi Štohlovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Dále Ing. Lukáši Honcovi a společnosti Acam Solution, díky kterým mohla tato diplomová práce vzniknout. V neposlední řadě chci poděkovat své přítelkyni a rodičům, za podporu a trpělivost.

V Brně dne: 1. června 2020 .....

podpis autora

Obsah

Seznam obrázků

Úvod

Automatizace procesů v moderním průmyslu je běžnou praxí. Automatizují se především jednoduché a opakující se úkony. Takovéto úkony však nenajdeme pouze u výrobních linek. Značná část práce programátorů PLC je opakující se, a ve své podstatě jednoduchá. Musí se založit projekt, vytvořit zařízení, navázat základní bloky a další objekty, které budou v projektu využívány. Ta nejdůležitější práce programátora nastává až po vytvoření těchto základních prvků, a to programování logiky a funkčnosti jednotlivých akčních členů. Pro firmu, zaměstnávající programátory, je žádoucí, aby nad základním definováním projektu strávil co nejméně času. Každá firma má, nebo by mít měla, svůj vzor programu, který uplatňuje ve svých projektech. Tím se ulehčí jak zaučení nových programátorů, tak vytváření a orientace v projektech nových. Pokud je tento vzor správně navržen, je možné generovat značnou část kódu automaticky. Na trhu existuje několik produktů, které umožňují generovat kód automaticky. Tyto software mají malou škálu možností, nutí firmy používat jejich vzor projektů a v neposlední řadě stojí nemalé peníze.

Tato práce se zabývá možností automatického generování kódu pro PLC od firmy Siemens. První část se zaměřuje na aktuálně dostupný software a nástroje pro automatické generování kódu a možnostmi C# knihovny Siemens TIA Openness. Dále je vytvořen návrh vzoru projektu v TIA portal, včetně struktury zařízení, bloků, tabulek tagů a uživatelsky definovaných datových typů. Nakonec je navržena a vytvořena uživatelská aplikace umožňující automatické generování kódu v TIA portal v15, dle předem vytvořené struktury programu, vhodná především pro větší projekty s rozdělením na jednotlivé stanice.

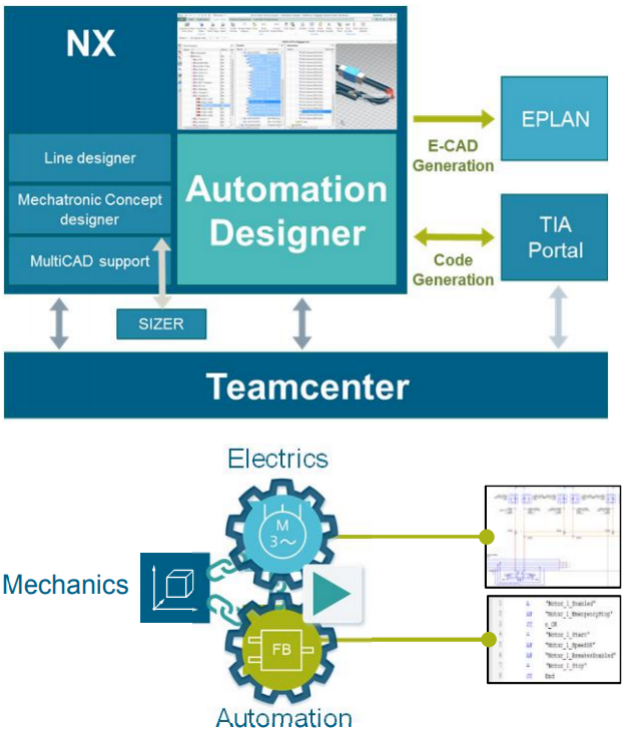
1 Přehled nástrojů pro generování kódu PLC Siemens

1.1 Automation Designer

Automation Designer je software od společnosti Siemens. Jedná se o inženýrskou platformu využívanou především pro zpracovatelský průmysl. Hlavním účelem tohoto softwaru je snížit čas při instalování nových zařízení nebo vytváření nové linky. Toho je docíleno opakovaným využíváním informací a možností generovat informace potřebné k implementaci zařízení nebo linky.

Automation Designer má tři hlavní funkce

- Funkční inženýrství založené na layoutu
- Generátor pro PLC kód, elektrické schéma EPLAN a WinCC flexible
- Datové spojení s dřívějšími inženýrskými fázemi, například mechanická simulace



Obr. 1.1: Schéma návrhu pomocí Automation Designer [2]

1.1.1 Generování PLC kódu

Automation Designer je schopný generovat PLC kód pro Siemens software SIMATIC STEP 7. Spolu s PLC kódem je schopný generovat také hardwarovou konfiguraci a tabulku symbolů pro daný projekt. Aby byl Automation Designer schopný generovat PLC kód, je zapotřebí zadat typ kódu a šablony pro generování. Toho je docíleno pomocí přidávání funkčních blokových diagramů do palety šablon. Uživatel si z vytvořených šablon poskládá vlastní projekt a Automation Designer následně vytvoří PLC program. Tento program je poté možné editovat v STEP 7. [1]

1.1.2 Generování HMI obrazovek

Automation Designer využívá XML formát pro importování a exportování dat z a do WinCC flexible. Tento software je schopný generovat celé HMI panely až do 4 různých úrovní detailu. Tagy a ostatní informace mohou být propojeny uvnitř Automation Designer k signálům z PLC struktury před vytvořením HMI obrazovek.

Aby byl software schopný generovat HMI obrazovky, potřebuje znát ikony a další informace pro generování každé vrstvy HMI. To je vyřešeno pomocí importu XML souborů do šablon v softwaru. Tyto XML soubory by měly obsahovat veškeré informace potřebné k generování obrazovek.

Pomocí 2D-Layout inženýringu v Automation Designeru generátor HMI obrazovek bude vědět, na kterém místě má být dané vizualizované zařízení.

1.2 Simulink PLC coder

Simulink PLC Coder od společnosti MathWorks je software, který generuje hardwarově nezávislý Structure Text (ST) a Ladder Diagram (LAD) z prostředí Simulink. ST a LAD bloky mohou být generovány pro rozhraní PLCopen XML, Siemens TIA Portal, Rockwell Automation Studio 5000 a mnoho dalších. [3]

1.2.1 Generování Structured Text

S podporou více než 180 simulinkových bloků, všemi Stateflow konstrukcemi a většinou funkcí knihovny MATLAB, Simulink PLC Coder generuje Structured Text z modelu řídícího systému, obsahující smyčky zpětných vazeb, stavovou logiku a matematické algoritmy.

```
15  FUNCTION_BLOCK SimpleSubsystem
16  VAR_INPUT
17      ssMethodType: SINT;
18      U: LREAL;
19  END_VAR
20  VAR_OUTPUT
21      Y: LREAL;
22  END_VAR
23  VAR
24      UnitDelay_DSTATE: LREAL;
25  END_VAR
26  VAR_TEMP
27      rtb_Gain: LREAL;
28  END_VAR
29  CASE ssMethodType OF
30      SS_INITIALIZE:
31          (* InitializeConditions for UnitDelay: '<S1>/Unit Delay'  *)
32          UnitDelay_DSTATE := 0;
33
34      SS_OUTPUT:
35          (* Gain: '<S1>/Gain' incorporates:
36           *  Inport: '<Root>/U'
37           *  Sum: '<S1>/Sum'
38           *  UnitDelay: '<S1>/Unit Delay'
39            *)
40          rtb_Gain := (U - UnitDelay_DSTATE) * 0.5;
41
42          (* Outport: '<Root>/Y'  *)
43          Y := rtb_Gain;
44
45          (* Update for UnitDelay: '<S1>/Unit Delay'  *)
46          UnitDelay_DSTATE := rtb_Gain;
47
48  END_CASE;
49  END FUNCTION BLOCK
```

Obr. 1.2: Generovaný ST kód pro CoDeSys Version 2.3 PLC IDE [3]

1.2.2 Import a generování Ladder diagramů

Pro systémy využívající Rockwell Automation Studio 5000 je možné importovat LAD kód přímo do Simulinku pro simulaci a verifikaci nebo generovat LAD kód z Simulinkových modelů pro Studio 5000.

1.2.3 Optimalizace kódu

Simulink PLC Coder dokáže pomocí dead-code eliminace, expression foldingu a znovu použití podsystémů optimalizovat vygenerovaný ST nebo LAD kód. Sníží se tak jeho velikost v paměti a zvýší rychlost provedení kódu.

1.2.4 Testování a verifikace kódu

Pomocí nástroje Simulink PLC Coder je možné generovat testy pro verifikaci shodnosti simulací provedených v Simulinku a výsledků z PLC.

```
        (* TEST CYCLE SETUP  *)
        cycle_In1 := tb_In1[testCycleNum];
        cycle_In2 := tb_In2[testCycleNum];
        cycle_In3 := tb_In3[testCycleNum];
        cycle_Out := tb_Out[testCycleNum];
        IF testCycleNum = 0 THEN
            (* INIT  *)
            PID_Subsystem(i0_PID_Subsystem, 0, cycle_In1, cycle_In2, cycle_In3, out_Out);
        END_IF;
        (* STEP  *)
        PID_Subsystem(i0_PID_Subsystem, 1, cycle_In1, cycle_In2, cycle_In3, out_Out);
        (* VERIFY  *)
        IF testVerify THEN
            IF cycle_Out = 0.0 THEN
                IF ABS(out_Out) > 9.9999997473787516E-5 THEN
                    testVerify := 0;
                END_IF;
            ELSIF ABS(out_Out - cycle_Out) > (9.9999997473787516E-5 * ABS(cycle_Out)) THEN
                testVerify := 0;
            END_IF;
        END_IF;
        testCycleNum := testCycleNum + 1;
    END_IF;
END_IF;
```

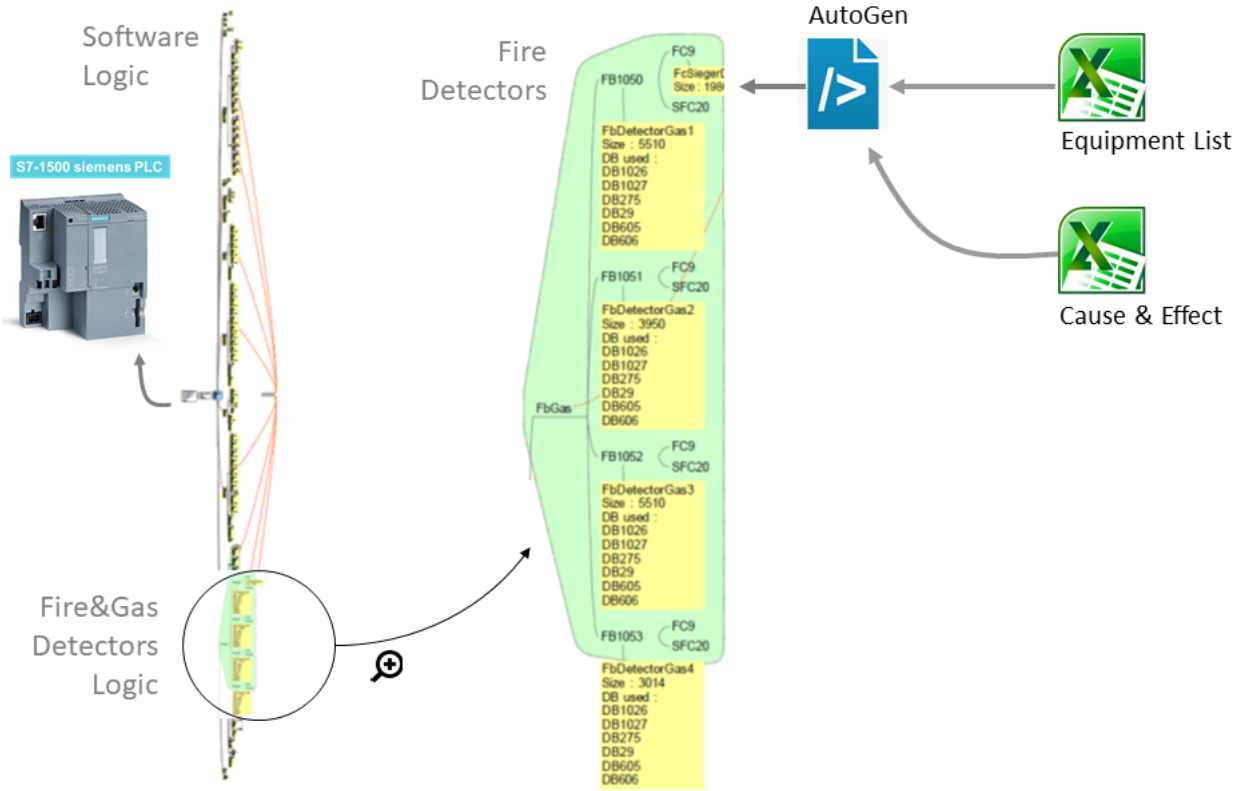
Obr. 1.3: Testovací kód pro verifikaci simulace [3]

1.3 AutoGen

Pomocí software společnosti Remote Intuitive Visual OperationS (RIVOPS) s názvem Autogen, je možné generovat procesní a řídící logiku ze seznamu vybavení. Pomocí python skriptů lze automatizovat základní komponenty výrobní linky včetně její údržby. [4]

AutoGen se skládá ze šesti částí:

- AUTOGEN for AlarmsGrouping - generování alarmů
- AUTOGEN for Logic - generování logiky
- AUTOGEN for OPC - generování OPC komunikace
- AUTOGEN for 3D - generování 3D vizualizace
- AUTOGEN for 2D - generování 2D vizualizace
- AUTOGEN for DWG - generování dokumentace



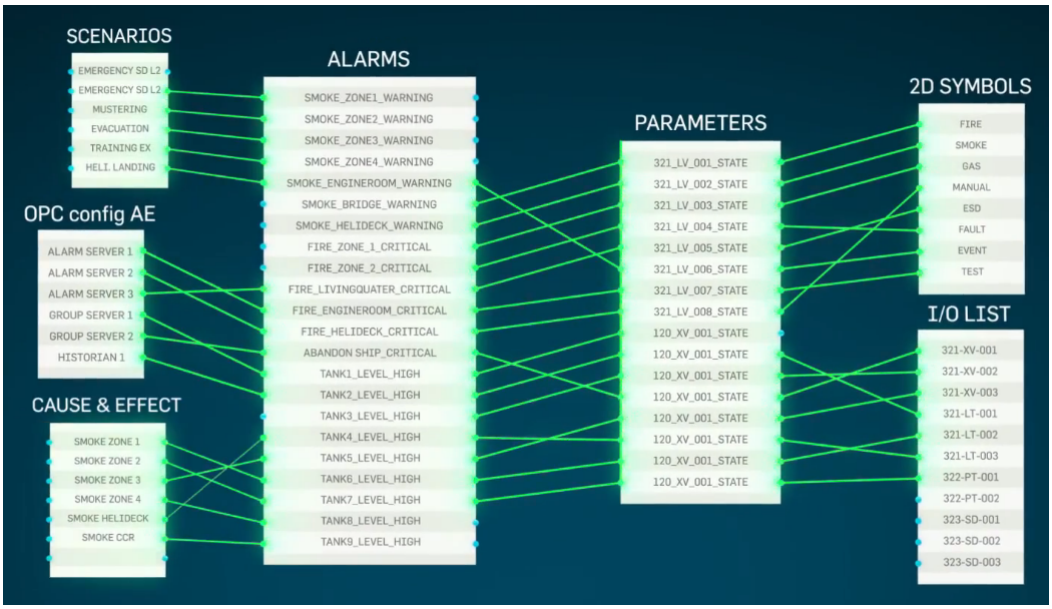
Obr. 1.4: AutoGen - Schéma generování kódu [4]

1.3.1 Generování kódu

AutoGen skripty jsou kompatibilní s většinou PLC integrovanými vývojovými prostředími (IDE). Skripty generují kód na základě normy IEC 61131:

- IL - instruction list
- ST - structured text
- LD - ladder diagram
- FBD - function block diagram

Pro generování kódu je potřeba znát veškerá pracovní prvky procesu (válce, motory, atp.). Tyto prvky jsou se svými parametry uloženy v Equipment Listu. Dále je potřeba nadefinovat Cause & Effect list. V tomto listu jsou nadefinovány jednotlivé interakce mezi danými prvky. AutoGen pomocí svých skriptů vytvoří potřebnou logiku pro PLC.



Obr. 1.5: AutoGen - Schéma procesu [4]

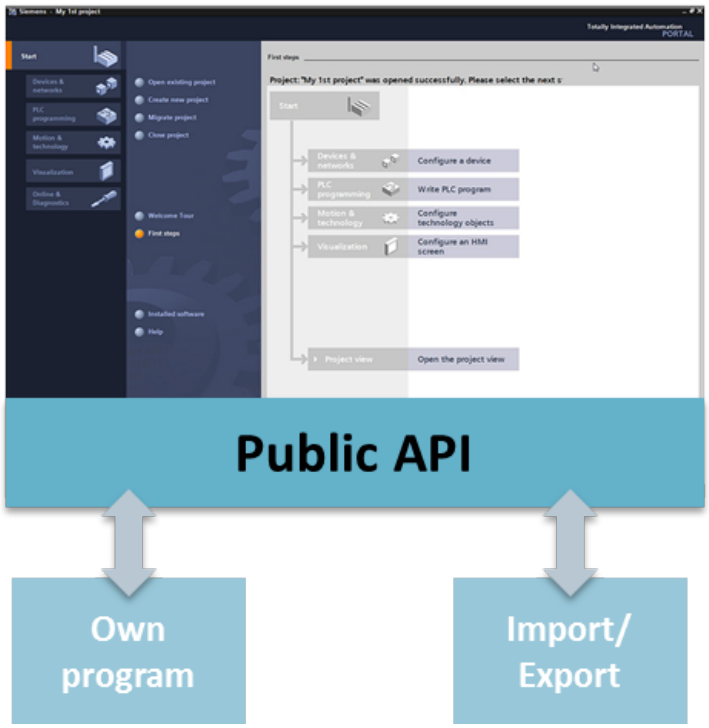
2 Siemens TIA Openness

Totally Integrated Automation Portal (TIA) Openness je API rozhraní pro WinCC a STEP 7, pomocí kterého lze integrovat TIA portal ve vlastním vývojovém prostředí a automatizovat dané procesy.

Pomocí externího vývojového prostředí lze vytvářet vlastní aplikace využívající funkce software TIA portal.

Hlavní využití této API je následující:

- Vytváření projektových dat
- Modifikace projektových dat
- Mazání projektových dat
- Čtení projektových dat
- Zpřístupnění projektových dat dalším aplikacím

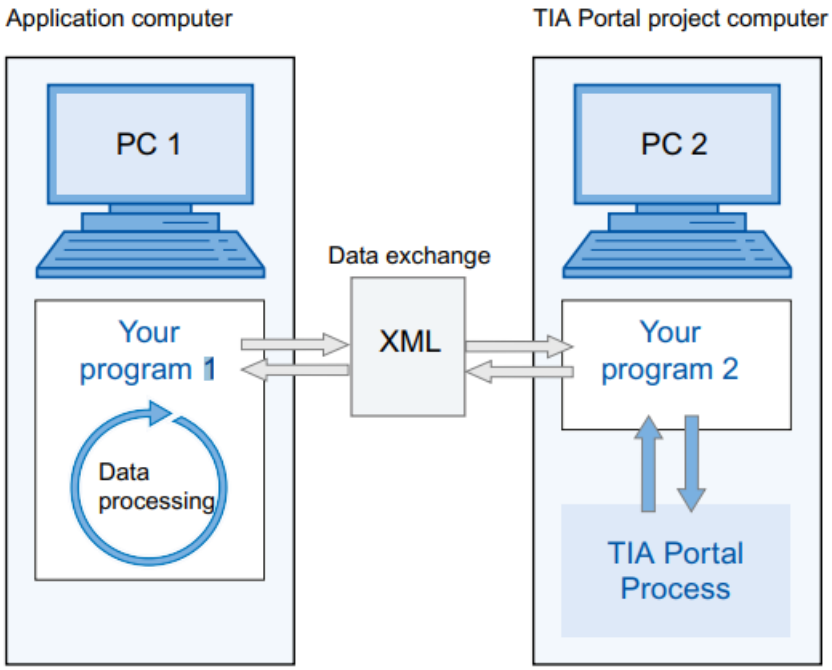


Obr. 2.1: Schéma funkce API TIA Openness [5]

2.1 Konfigurace TIA Openness

2.1.1 Aplikace a TIA Portal beží na rozdílných PC

Výměna dat probíhá pomocí XML souboru. Tento XML soubor může být exportován nebo importován do naších programů. Data exportovaná z TIA Portal projektu do PC2 mohou být modifikována v PC1 a znovu naimportována (jak lze vidět na obrázku). Lze archivovat vyměněné soubory pro verifikační účel. Výměna dat může probíhat v rozdílných místech a časech. [5]

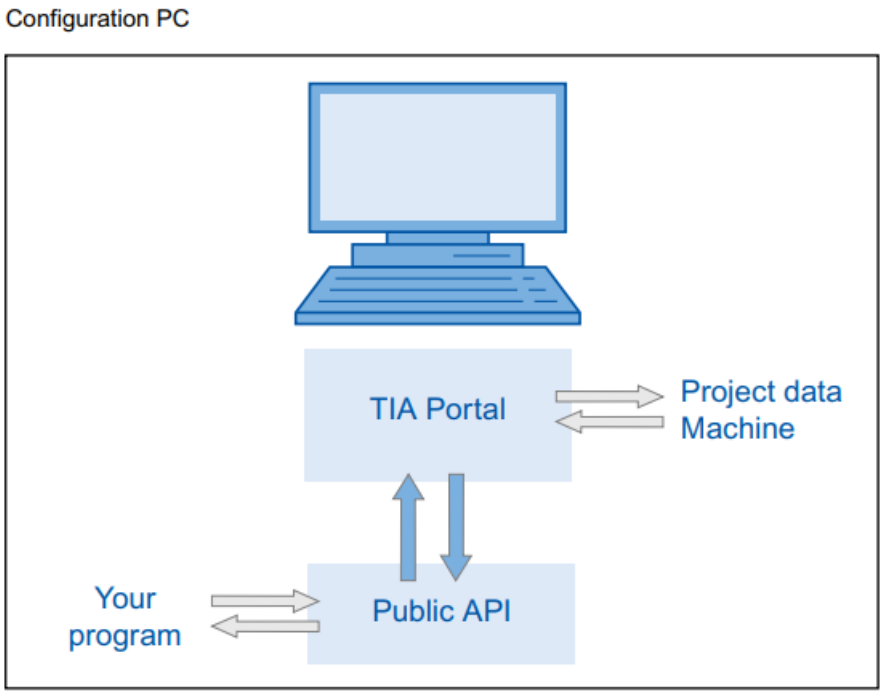


Obr. 2.2: Schéma konfigurace pro 2 PC [5]

2.1.2 Aplikace a TIA Portal beží na jednom PC

Naše aplikace běží zároveň s aplikací TIA Portal (lze nastavit běh bez vizuálního rozhraní), má veškerý přístup k TIA Portal aplikaci a lze provádět všechny potřebné funkce.

Tato konfigurace bude využita pro mou aplikaci. Na jednom stroji tedy poběží jak aplikace TIA Portal, tak zároveň mnou vyvinutá aplikace komunikující s TIA Portal.



Obr. 2.3: Schéma konfigurace pro 1 PC [5]

2.2 Základní metody API TIA Openness pro manipulaci s projektem

V této části jsou popsány nejdůležitější metody API, které jsou využívány pro manipulaci s projektem.

2.2.1 Start TIA Portal

Pomocí této metody lze přes API spustit aplikaci TIA Portal. Není tedy nutné zvlášť spouštět vytvořenou aplikaci a TIA Portal.

```
public void StartTIA(bool withInterface)
        {
            if (withInterface == false)
            {
                MyTiaPortal = new
                   TiaPortal(TiaPortalMode.WithoutUI);
                _tiaProcess = TiaPortal.GetProcesses()[0];
            }
            else
                MyTiaPortal = new
                   TiaPortal(TiaPortalMode.WithUI);
        }
```

2.2.2 Vypnutí TIA Portal

Metoda pro vypnutí aplikace TIA Portal. Využívaná pro ukončení procesu aplikace TIA Portal.

```
public void DisposeTIA()
        {
            MyTiaPortal.Dispose();
        }
```

2.2.3 Vytvoření projektu

Metoda, pomocí které lze vytvořit nový projekt v rozhraní TIA Portal pomocí API. Je potřeba zadat jméno projektu a cílovou složku, ve které se projekt vytvoří.

```
public void CreateProject(string projectName, string
   path = null)
        {
            path = ProjectPath + "Projects";
            ProjectComposition projectComposition =
               MyTiaPortal.Projects;
            DirectoryInfo targetDirectory = new
               DirectoryInfo(path);
            MyProject =
               projectComposition.Create(targetDirectory,
               projectName);
        }
```

2.2.4 Otevření projektu

Metoda, pomocí které lze otevřít projekt v rozhraní TIA Portal pomocí API. Je potřeba zadat adresářovou cestu k žádanému projektu.

```
public void OpenProject(string projectPath)
        {
            MyProject = MyTiaPortal.Projects.Open(new
               FileInfo(projectPath));
        }
```

2.2.5 Uložení a zavření projektu

Metody, pomocí kterých lze uložit a zavřít projekt v rozhraní TIA Portal.

```
public void SaveProject()
        {
            MyProject.Save();
        }
public void CloseProject()
        {
            MyProject.Close();
        }
```

2.2.6 Připojení k projektu

Pomocí této metody se lze připojit k již otevřené instanci TIA Portal a k danému projektu, který je v této instanci otevřen.

```
public void ConnectToProject()
        {
            IList<TiaPortalProcess> processes =
               TiaPortal.GetProcesses();
            switch (processes.Count)
            {
                case 1:
                    _tiaProcess = processes[0];
                    MyTiaPortal = _tiaProcess.Attach();
                    MyProject = MyTiaPortal.Projects[0];
                    break;
                case 0:
                    return;
                default:
                    return;
            }
        }
```

2.3 Vytváření projektu pomocí API Openness

Pokud jsme úspěšně připojeni k instanci TIA Portal, na které je otevřen projekt, můžeme tento projekt zcela ovládat a modifikovat. Ve většině případů projekt modifikujeme pomocí importu daných prvků (bloky, tagy, atd.)

2.3.1 Hardwarová konfigurace

Pokud nám stačí jednoduchá konfigurace obsahující pouze jedno zařízení bez periferií, postačí využít metodu AddDevice. Tato metoda vyžaduje parametry název zařízení, katalogové číslo a verzi firmware.

Ve většině případů je však potřeba vytvořit rozsáhlou hardwarovou konfiguraci s větším počtem zařízení a rozsáhlou sítí. Toho docílíme pomocí konfiguračního souboru AML. Soubor typu AML je svou strukturou velmi podobný XML souborům. Společnost Siemens využívá typ AML pro export a import hardwarových konfigurací.

Tento konfigurační soubor získáme pomocí aplikace TIA Selection Tool, což je software od společnosti Siemens umožňující vytváření libovolných hardwarových konfigurací a jejich následného exportu do formátu AML.

Dalším způsobem, jak tento konfigurační soubor získat, je ručním nastavením hardwarové konfigurace ve vlastním projektu TIA Portal a jejím následném exportu. Takto exportovaný soubor je rovněž ve formátu AML a lze jej tedy použít na nahrání hardwarové konfigurace pomocí metody SetHWConfig, která vyžaduje parametry název konfigurace a adresářovou cestu k souboru.

```
<InternalElement ID="c2dff7c2-d0ed-4a7c-836b-754a612fe805" Name="ET 200SP station_1">
  <Attribute Name="TypeIdentifier" AttributeDataType="xs:string">
    <Value>System:Device.ET200SP</Value>
  </Attribute>
  <InternalElement ID="7be1cb45-74f5-486e-b9ce-d07f79018b9d" Name="Rack_0">
    <Attribute Name="TypeName" AttributeDataType="xs:string">
      <Value>Rack</Value>
    </Attribute>
    <Attribute Name="PositionNumber" AttributeDataType="xs:int">
      <Value>0</Value>
    </Attribute>
    <Attribute Name="BuiltIn" AttributeDataType="xs:boolean">
    <Attribute Name="TypeIdentifier" AttributeDataType="xs:string">
    <InternalElement ID="f84af001-1d56-4929-b009-9d61b7cc0e05" Name="+RH1-K10">
    <InternalElement ID="87d56a21-4a07-4f9a-9e61-1b24b7a1a9dd" Name="Server module 1">
```

Obr. 2.4: Část AML kódu

2.3.2 Bloky

Veškeré organizační, funkční a datové bloky lze importovat a exportovat. Pro export je využívána metoda GetBlock, pomocí které lze po zadání názvu zařízení, bloku a skupiny exportovat daný blok ve formátu XML.

```
public void GetBlock(string deviceName, string
    blockName, string groupName = null, string path =
    null)
```

Stejným způsobem funguje metoda AddBlock, která daný blok importuje do našeho projektu.

```
public void AddBlock(string deviceName, string
    blockName, string groupName = null, string path =
    null)
```

2.3.3 Tagy

Stejně jako bloky jdou exportovat a importovat také tabulky tagů. Výsledný exportovaný soubor je rovněž ve formátu XML. Export tabulek tagů zajišťuje metoda GetPlcTagTable s parametry název zařízení a název tabulky.

```
 public void GetPlcTagTable(string deviceName, string
     nameTable = null, string path = null)
```

Import zajišťuje metoda AddPlctagTable.

```
public void AddPlcTagTable(string deviceName, string
    nameTable = null, string path = null)
```

2.3.4 UDT

Exportovat a importovat lze také User Defined Type (UDT). Výsledný exportovaný soubor je rovněž ve formátu XML. Export UDT zajišťuje metoda GetUDT s parametry název zařízení a název UDT.

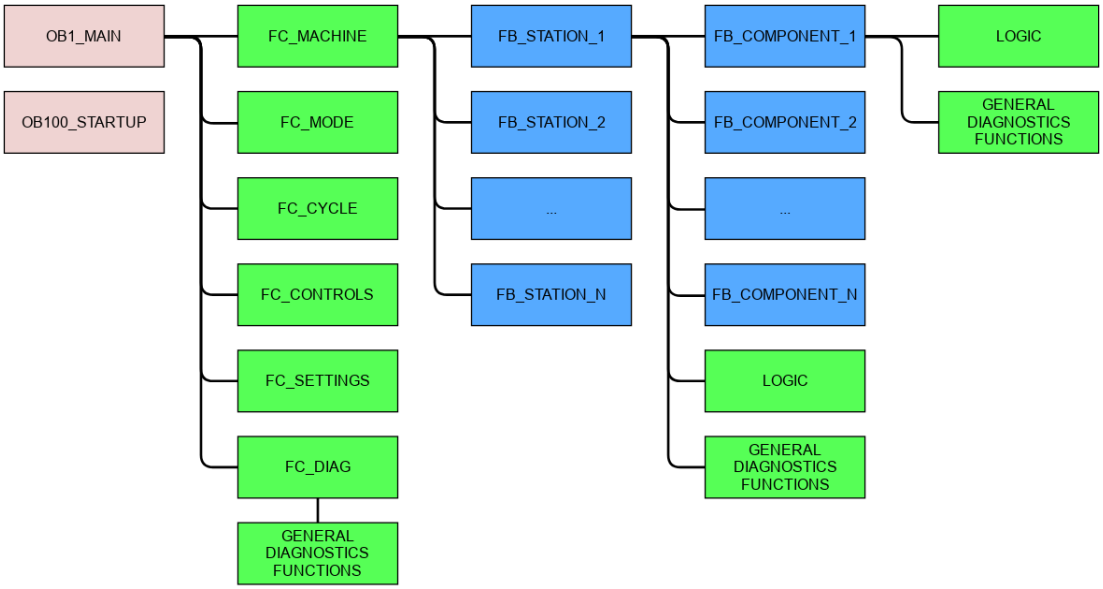
```
 public void GetUDT(string deviceName, string nameTable
     = null, string path = null)
```

Import zajišťuje metoda AddUDT.

```
public void AddUDT(string deviceName, string nameTable =
    null, string path = null)
```

3 Návrh struktury projektu v TIA portal

Struktura projektu se skládá z čtyř částí. První část definuje organizační bloky. Z těchto bloků jsou volány veškeré ovládací funkce a bloky potřebné pro chod stroje. Druhou částí jsou základní funkce. Účel jednotlivých funkcí je vysvětlen dále. Třetí částí struktury projektu jsou stanice. Ty zaštiťují jednotlivé části stroje (zásobníková stanice, měřící stanice) a obsahují veškerou potřebnou logiku pro chod dané stanice. Čtvrtou částí jsou jednotlivé komponenty stanic, jako například ventily a motory.



Obr. 3.1: Schéma návrhu projektu v TIA Portal

3.1 Organizační bloky a základní funkce

3.1.1 OB1_MAIN a Diagnostické OB

Hlavním organizačním blokem je OB1_MAIN, z kterého jsou volány veškeré ostatní bloky a funkce. Pro co největší přehlednost programu je OB1_MAIN určen pouze pro volání vybraných bloků a funkcí a nenachází se zde žádná přímá logika.

Z diagnostických důvodů je také potřeba do programu zahrnout veškeré diagnostické OB, jako například Diagnostic error interrupt[OB82], IO access error[OB122], Programming error[OB121] a další.

3.1.2 Základní řídicí funkce

V této části jsou popsány základní funkce, které obsahují řídicí logiku stroje. Všechny tyto funkce jsou volány přímo v OB1_MAIN.

FC_CONTROLS - Funkce obsahuje logiku obecných ovládacích prvků. Například zde můžeme zařadit ovládání majáku a výstražné sirény. Značení povolení přístupu do stroje nebo ovládání dalšího informačního značení.

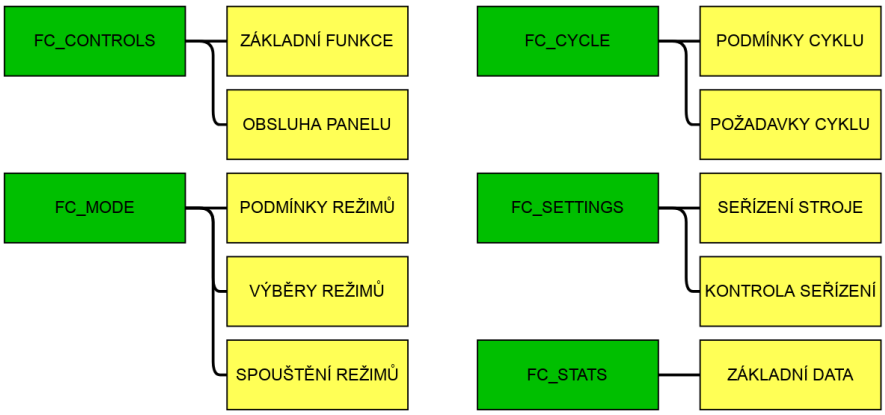
FC_CYCLE - V této funkci je řešeno ovládání cyklu stoje. Cyklus je sekvence kroků, které stroj automaticky vykonává, probíhá v automatickém režimu. Tato funkce obsahuje podmínky pro spuštění, ukončení a řízení cyklu.

FC_MODE - Ovládání režimu stroje. Funkce obsahuje podmínky připravenosti spuštění manuálního a automatického režimu, podmínky připravenosti ukončení manuálního a automatického režimu, výběr manuálního nebo automatického režimu, požadavek spuštění a ukončení režimu a kontrola spuštěného režimu.

FC_STATS - Tato funkce obsahuje práci se základními daty jako je manipulace s časem PLC, časy cyklu, nebo LiveByte (byte využívaný pro kontrolu běhu PLC ve vizualizaci).

FC_SETTINGS - Funkce řešící seřízení stroje. V některých případech je potřeba dostat stroj do výchozí polohy. Tomu se rozumí například najetí motorů do požadovaných poloh, sepnutí příslušných ventilů, válců, atd. Jakmile je seřízení hotovo, lze stroj uvést do automatického módu a sepnout cyklus (za předpokladu splnění všech ostatních podmínek).

FC_MACHINE - V této funkci jsou volány veškeré funkční bloky stanic, které stroj obsahuje.



Obr. 3.2: Schéma základních funkcí

3.2 Datové bloky

Při programování PLC existují dva typy datových bloků, globální a instanční. Instanční datové bloky jsou využívány funkčními bloky a jejich proměnné jsou definovány daným funkčním blokem. V globálních datových blocích jsou uloženy globální proměnné, s kterými lze pracovat v celém programu. V této části budou popsány využívané globální datové bloky.

DB_DATA - V tomto datovém bloku jsou uložena veškerá data týkající se vyráběného produktu. Proměnné tohoto datového bloku jsou definovány uživatelsky definovaným typem PRODUCT_GROUP.

DB_HMI - Proměnné v tomto bloku slouží ke komunikaci mezi PLC a HMI (Human-Machine Interface). Skládají se z pěti základních typů proměnných a jedná se o požadavky z HMI. Proměnné typu BASE jsou obecné proměnné, jako je potvrzení alarmů, livebyte, watchdog. Proměnné typu MODE jsou využívány pro ovládání režimů stroje (manuální, automatický). Proměnné typu CYCLE ovládají cyklus stroje (start, stop cyklu). Proměnné typu SETTINGS jsou využívány pro seřizování stroje a obsahují start, stop seřízení a seřízení hotovo. Nakonec proměnné OTHERS, kde jsou zahrnuty ostatní potřebné proměnné pro komunikaci s HMI, například pro ovládání některého z modulů.

DB_RECIPE - Datový blok RECIPE obsahuje proměnnou Recipe, která je definována UDT_RECIPE. Ten je popsán dále v části s uživatelsky definovanými typy. Proměnná je typu retain, což znamená, že se ukládá do vnitřní paměti PLC a neztratí se tak nastavené hodnoty ani po vypnutí PLC.

DB_RETAIN - Tento datový blok obsahuje veškeré ostatní retainové proměnné mimo receptury, které jsou potřeba v programu. Může se jednat například o nastavené pozice motorů.

DB_SYS - Datový blok DB_SYS se skládá z pěti typů proměnných. První jsou proměnné SAFETY, které značí, zda je v pořádku bezpečnost (zamknuté bezpečnostní zámky, stroj je ve stopu). Jako druhý typ proměnných jsou DATA. Mezi tyto proměnné patří aktuální čas, čas cyklu, maximální čas cyklu a minimální čas cyklu. Třetí typ proměnných je MODE. Tyto proměnné řídí režim stroje. Proměnné CYCLE řídí cyklus stroje a proměnné SETTINGS řídí seřizování stroje.

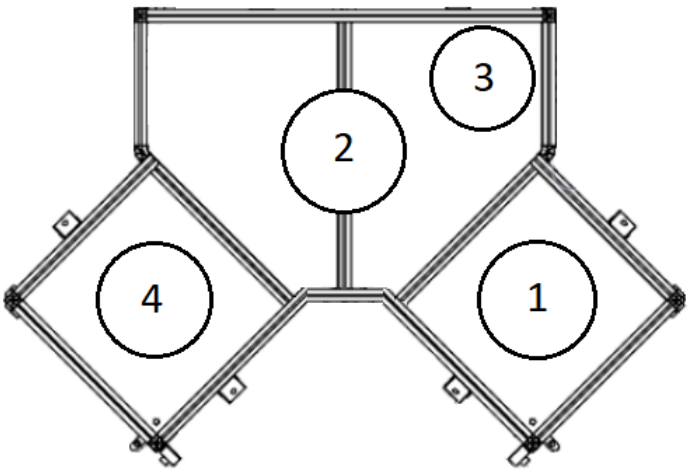
3.3 Stanice

Z důvodu škálovatelnosti a přehlednosti programu, bude stroj vždy rozdělen na jednotlivé stanice.

Například u jednoduchého jednoúčelový stroje pro automatickou výstupní kontrolu dílů, kde z zakládacího šuplíku odebírá díly robot, změří se jeho průměr a následně se vyhodnotí, zda se jedná o OK nebo NOK díl a založí se tak do příslušného výstupního boxu, můžeme takovýto stroj rozdělit do čtyř stanic. První stanice bude zakládací. Druhou stanicí bude robot. Třetí bude měřící stanice a poslední stanicí jsou výstupní boxy. Každá z těchto stanic obsahuje své specifické komponenty a svou vlastní logiku. Pomocí tohoto rozdělení na stanice, je možné jednoduše nahrazovat a přidávat jednotlivé stanice. Kdyby byla potřeba místo měření průměru, dílek ofukovat, můžeme navrhnout ofukovou stanici a v programu jen nahradit funkční bloky pro příslušné stanice.

Funkční bloky stanic (FB_STATION) jsou volány z FC_MACHINE. Jak již bylo vysvětleno výše, bloky obsahují veškerou logiku pro řízení stroje. Každý funkční blok vytváří také vlastní instanční datový blok. Tyto datové bloky obsahují vstupní, výstupní, dočasné a statické proměnné.

1. Vstupní stanice
2. Robot
3. Měřící stanice
4. Výstupní stanice



Obr. 3.3: Layout stroje rozděleného na stanice

3.3.1 Datová struktura funkčního bloku

Vstupy lze rozdělit na pět typů:

1. Obecné vstupy
   Tyto vstupy slouží pro základní ovládání stanice. Veškeré stanice obsahují tyto vstupy.
   - I_Auto - Značí zda je zapnut automatický režim stroje
   - I_Man - Značí zda je zapnut manuální režim stroje
   - I_Safety - Bezpečnost v pořádku
   - I_Enable - Povolení stanice
   - I_Quit - Potvrzení poruch
   - I_Reset - Resetování
   - I_Data_Product - Data produktu postupující strojem

2. Vstupy blokací
   Blokační vstupy pro povolování přístupu do stroje jak z pozice obsluhy, tak z pozice např. robota, který do stanice zasahuje.
   - I_Input_Block - Značí zda je požadavek na blokaci vstupu
   - I_Input_Block_Condition - Podmínky vstupní blokace
   - I_Input_ACK - Potvrzení vstupu
   - I_Input_ACK_Condition - Podmínka potvrzení vstupu
   - I_Output_Block - Značí zda je požadavek na blokaci výstupu
   - I_Output_Block_Condition - Podmínky výstupní blokace
   - I_Output_ACK - Potvrzení výstupu
   - I_Output_ACK_Condition - Podmínka potvrzení výstupu

3. Konfigurační vstupy
   Konfigurační vstupy slouží pro nastavení prvků ve stanici. Na rozdíl od obecných vstupů má každá stanice vlastní konfigurační vstupy. Například pro šuplíkovou stanici můžeme za konfigurační vstupy považovat počet šuplíků s počtem řad a sloupců, ze kterých bude díl odebírán.

4. Příkazové vstupy
   Příkazové vstupy jsou využívány u stanic, které jsou ovládány stanicemi jinými, nebo přímo jednotlivými vstupy PLC. Pokud stanice takto ovládána není, má vnitřní příkazovou strukturu (CMD) využívanou pro ovládání příslušných komponent. Vztaženo znovu na šuplíkovou stanici, za příkazové vstupy bychom považovali vstup odemknout šuplík (Reálné tlačítko, tudíž přímý vstup do PLC).

5. Zpětnovazební vstupy
   Zpětnovazební vstupy obsahují informace senzorů, například indukční senzor značící úspěšné zavření ventilu.

6. Diagnostické vstupy
   Tyto vstupy má každá stanice stejné. Definují offset v alarmovém DB, číslo alarmového DB a alarmovou proměnnou.

Výstupy lze rozdělit na dva typy:

1. Obecné vstupy
   Jedná se o základní výstupy stanice. Veškeré stanice obsahují tyto výstupy.
   - Q_Input_Block - Značí zda je vstup blokován
   - Q_Output_Block - Značí zda je výstup blokován
   - Q_Product - Výstupní produkt ze stanice

2. Definovatelné výstupy
   Tyto výstupy jsou definovány dle požadavků na stanici. Pro šuplíkovou stanici budou definovány výstupy pro zamykání válců. Pro stanici, která obsahuje motor bude jeden z výstupů definovat spuštění motoru.

Statické a dočasné proměnné

Statické proměnné uchovávají data, se kterými stanice aktuálně pracuje, jedná se například o data produktu nebo stavového automatu. Dočasné proměnné jsou využívány na pomocné výpočty.

Konstanty

Jako konstanty jsou uloženy čísla jednotlivých kroků stavového automatu. V programu se pak lze na jednotlivé kroky odkazovat jejími názvy a ne čísly. Dále jsou tímto způsobem zadefinovány jakékoliv proměnné, které se v celém cyklu nemění.

3.3.2 Vnitřní struktura funkčního bloku

Každý funkční blok stanic se skládá z těchto základních částí:

1. Systém
   V této části kódu se nachází inicializace proměnných, pro správnou funkci stavového automatu, nebo jiné logiky ve stanici.

2. Blokace
   Pokud je stanice v blokaci, znamená to, že je v ní aktuálně vykonáván proces jiným zařízením nebo obsluhou. Například pro šuplíkovou stanici bude vstupní blokace provedena obsluhou. V tomto stavu nesmí žádné jiné zařízení zasáhnout do této stanice. Výstupní blokace je prováděna ostatními zařízeními (stanicemi). Pokud je aktivní výstupní blokace, není možný přístup pro obsluhu.

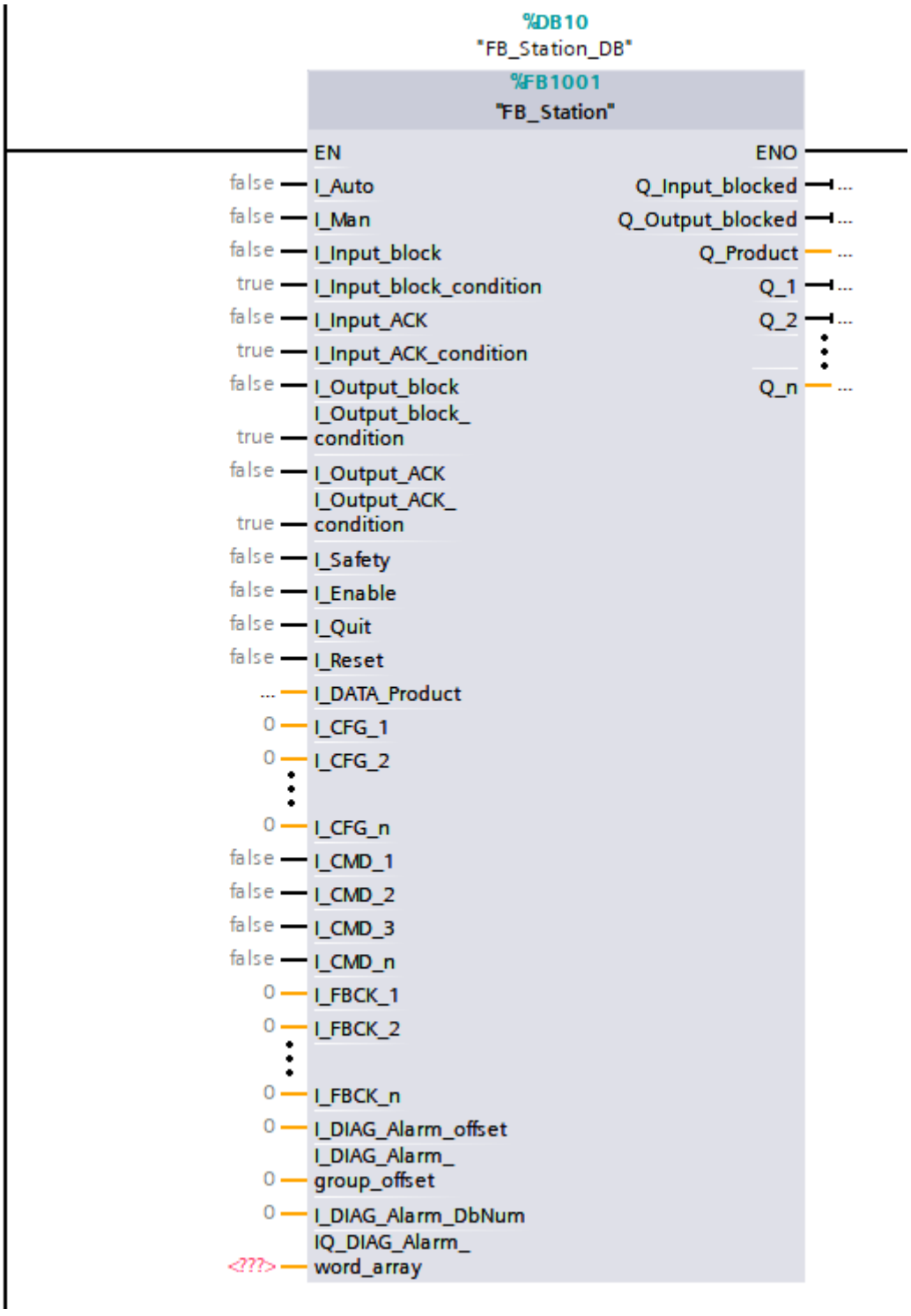
3. Data
   Manipulace s daty a jejich předávání ze vstupu na výstup po provedení cyklu stanice.

4. Cyklus
   V této části kódu je naprogramováno chování stanice v automatickém režimu, v cyklu, většinou pomocí stavového automatu.

5. Ovládání hardwarových komponent
   Pro přehlednost není ovládání jednotlivých akčních členů zakomponováno přímo do stavového automatu, ale je umístěno pouze na jednom místě. Tím pádem, pokud bude potřeba spínat ventil na více místech stavového automatu, přivedou se dané stavy, ve kterých je potřeba ventil spínat, na vstup spínání v části ovládání hardwarových komponent. Takto bude veškeré ovládání daného akčního členu na jednom místě a není potřeba prohledávat stavový automat.

6. Hardwarové komponenty
   Jsou zde volány funkční bloky hardwarových komponent, které jsou využívány pro řízení daných akčních členů. Pro každý typ akčního členu je definován jedinečný funkční blok, tyto bloky jsou popsány dále.

7. Diagnostika
   Jsou zde definovány alarmy, které mohou ve stanici vzniknout a volány potřebné diagnostické funkce.



Obr. 3.4: Obecný funkční blok stanice

3.4 Komponenty

Komponenty se rozumí jednotlivé akční členy stroje. Každý komponent má vytvořený vlastní funkční blok, ve kterém je řešeno jeho ovládání. Funkční bloky jsou pojmenovány FB_NAZEV_KOMPONENTY. Mezi základní komponenty se řadí ventil(valve) a motor.

3.4.1 Datová struktura funkčního bloku

Vstupy lze rozdělit na 4 typy:

1. Obecné vstupy
   Tyto vstupy slouží pro základní ovládání komponenty. Veškeré komponenty obsahují tyto vstupy.
   - I_Auto - Značí zda je zapnut automatický režim stroje
   - I_Man - Značí zda je zapnut manuální režim stroje
   - I_Quit - Potvrzení poruchy
   - I_Enable - Povolení komponenty
   - I_Safety - Bezpečnost stroje
2. Konfigurovatelné vstupy
   Konfigurovatelné vstupy jsou jedinečné pro každý druh komponenty. Jedná se o veškeré potřebné vstupní signály, které jsou využívány danou komponentou.

3. Diagnostické vstupy
   Tyto vstupy má každá komponenta stejné. Definují offset v alarmovém DB, číslo alarmového DB a alarmovou proměnnou.

4. Vizualizační vstupy Vstupy vizualizačních proměnných se využívají pro řízení komponenty z vizualizace. Jedná se o uživatelsky definovaný typ pro danou komponentu. Například pro válec bude obsahovat vstupy jako otevřít, zavřít a časový limit zavření a otevření.

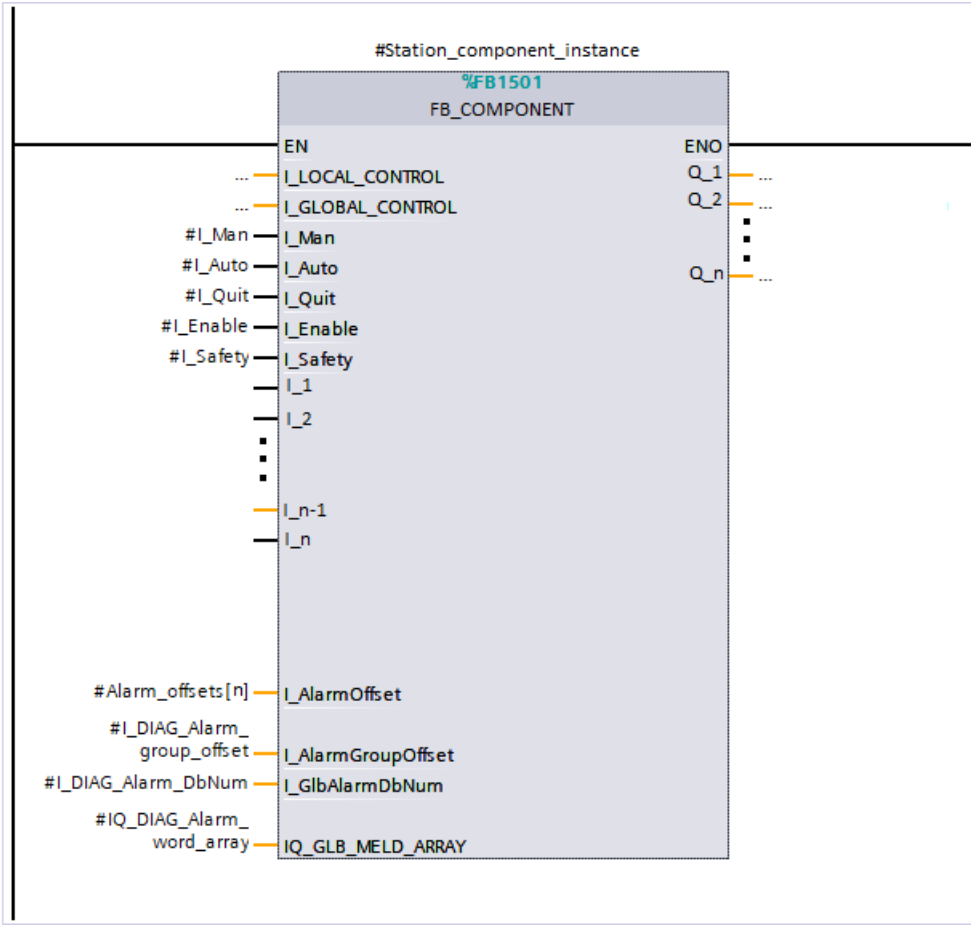
Výstupy komponent

Výstupy jednotlivých komponent se liší. Zpravidla se jedná o řídicí výstupy (spouštění motoru, otevírání a zavírání ventilu) a signalizační výstupy (porucha, v pozici).

3.4.2 Vnitřní struktura funkčního bloku

Každý funkční blok stanic se skládá z těchto základních částí:

1. Ovládání
   Tato část kódu obsahuje řídicí logiku dané komponenty, jako otevřít ventil, zapnout motor, referování motoru.
2. Výstupy
   Pomocí výstupů se spínají reálné signály, pomocí kterých jsou spouštěny akční členy.
3. Diagnostika
   Jsou zde definovány alarmy, které mohou v komponentě vzniknout a volány potřebné diagnostické funkce.
4. Status
   Status dané komponenty. Využívá se pro zjištění aktuálního stavu komponenty. U ventilu například statusy uzavřeno, otevřeno, porucha.

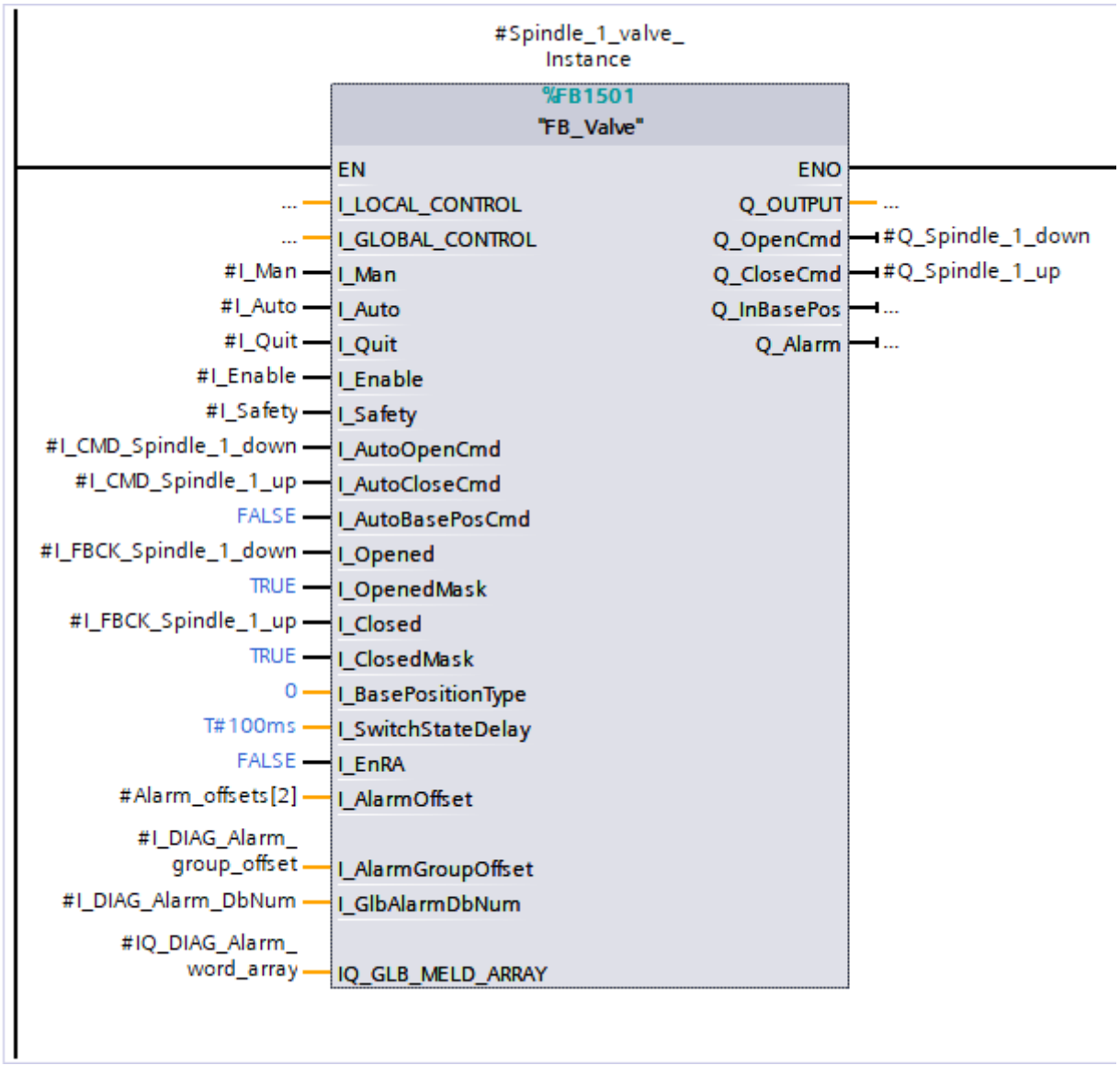


Obr. 3.5: Obecný funkční blok komponenty

3.4.3 Příklad komponenty - Ventil

Na následujícím obrázku je vytvořena komponenta ventil. U komponenty typu ventil jsou vytvořeny vstupy pro příkazy automatického otevření a zavření, zpětné vazby otevření a zavření.

Výstupy jsou signály otevření a zavření ventilu, signalizace poruchy a signalizace výchozí pozice.



Obr. 3.6: Funkční blok komponenty ventil

3.5 Tabulky tagů

Tabulky tagů jsou rozděleny na pět základních typů:

1. Alarm_offsets
   Tato tabulka je využívána pro definování konstant alarmových offsetů využívaných u jednotlivých stanic. Pomocí takového definování je následné vyhledávání alarmů velmi zjednodušeno.
2. Merkers
   Tabulka obsahující veškeré využívané merkery. Pro každý stroj bude tabulka obsahovat tzv. system and clock memory merkery, které zahrnují například alwaysTrue, alwaysFalse, clock_Hz.
3. System_Constants
   Definování systémových konstant jako je například číslo alarmového DB, stavové hodnoty diagnostických funkcí. Tyto konstanty jsou využívány převážně diagnostickými funkcemi.
4. User_Constants
   Uživatelsky definované konstanty jsou jedinečné pro každý stroj. Využívají se například pro definování parametrů stanic a produktu.
5. IO
   Tabulky obsahující vstupy a výstupy PLC. Jedinečné pro každý stroj.

3.6 Uživatelsky definované datové typy

Uživatelsky definované datové typy jsou rozděleny na čtyři základní druhy:

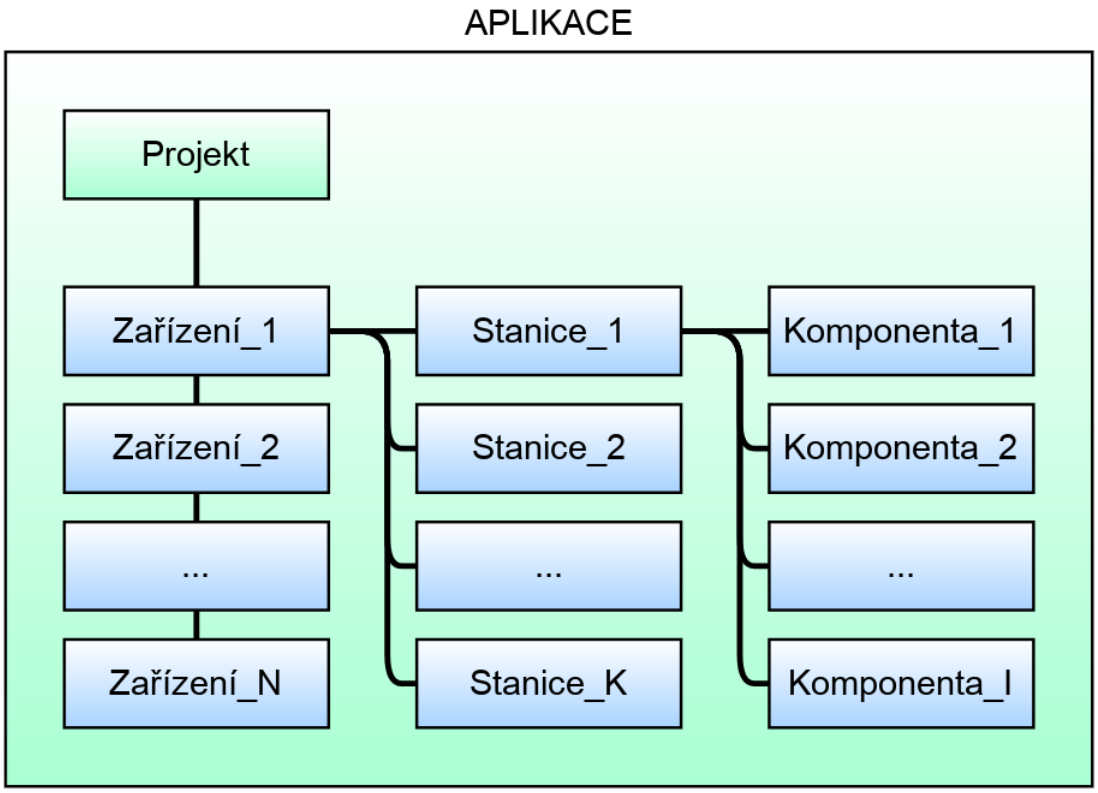
1. UDT_PRODUCT
   Toto UDT obsahuje proměnné pro ukládání základních informací o produktu, který bude strojem vyráběn. Jedná se o informace jako jsou naměřené hodnoty dílu nebo jeho identifikační číslo.
2. UDT_RECIPE
   UDT receptura definuje proměnné pro vytváření receptur. Obsahuje proměnné pro název receptury a její parametry.
3. UDT_VISU
   UDT definující proměnné pro ovládání jednotlivých komponent stroje z vizualizace. Obsahují statusy dané komponenty. U ventilu například statusy uzavřeno, otevřeno, porucha.

4 Návrh uživatelské aplikace

Uživatelská aplikace bude programována v jazyce C#. Pomocí tohoto jazyku lze vytvořit logickou část kódu (backend). Uživatelský interface (frontend) je programován jazykem XAML a bude vytvořen pomocí frameworku WPF. API TIA Openness je napsána také v jazyce C# a implementace potřebných metod pro manipulaci s projektem tak bude jednoduší.

V aplikaci bude možné vytvořit projekt, ke kterému se přiřadí zvolená zařízení. K těmto zařízením lze následně vybrat stanice s komponenty. Projekt můžeme ukládat a později znovu otevírat pro potřebné úpravy. Po nastavení veškerých parametrů se pomocí TIA Openness vytvoří projekt v TIA Portal V15.

Aplikace bude programována pomocí návrhového vzoru Model-View-ViewModel (MVVM) a .NET frameworku Windows Presentation Foundation (WPF)



Obr. 4.1: Schéma vytváření projektu pomocí aplikace

4.1 Windows Presentation Foundation

Je framework pro tvorbu komplexních formulářových aplikací, který je součástí .NET frameworku od verze 3.0. Disponuje širokou paletou formulářových prvků a také umožňuje téměř úplné editování vzhledu aplikace. Framework nabízí velkou škálu komponent, ze kterých lze formulář jednoduše sestavit. Jedná se o tlačítka, pole, posuvníky, popisky a další komponenty, takzvané controls (ovládací prvky). Důležitou vlastností je možnost vytvářet vlastní ovládací prvky. WPF se programuje ve speciálním jazyce XAML.

Používání vykreslovacího rozhraní, využívaného windows forms, Windows (GDI) je pomalé a přináší s sebou mnoho omezení. WPF využívá pro vykreslování `Direct3D`. To je rozhraní pro akcelerovanou grafiku. WPF aplikace jsou tedy plynulejší a méně zatěžují procesor. Díky nezávislosti na GDI je možné vytvářet graficky plně editovatelné aplikace. Lze například vkládat obrázky do tlačítek, cokoli zprůhlednit, aplikace skinovat (vytvářet je v určitém stylu) nebo animovat pomocí tzv. storyboardů. [6]

4.2 Návrhový vzor Model-View-ViewModel

Jedná se o návrhový vzor pro WPF aplikace a řeší oddělení logiky od uživatelského rozhraní. Tím se docílí zkrácení kódu, větší přehlednosti a lepší škálovatelnosti. MVVM odděluje data, stav aplikace a uživatelské rozhraní. Samotné WPF bylo vytvořeno tak, aby se v něm MVVM používal pohodlně. Proto se v něm využívá binding a command, který nahradil uživatelské rozhraní řízené událostmi.

Hlavní myšlenka MVVM je vytvořit třídu, která si drží aktuální stav aplikace. Tato třída se nazývá ViewModel. Té se dotazuje uživatelské rozhraní, které podle ní vykresluje ovládací prvky. A naopak zadá-li uživatel do uživatelského rozhraní nějaké údaje, přenesou se automaticky do ViewModelu. WPF je pro toto použití dobře uzpůsobeno, protože pomocí bindingu lze deklarativně napojit uživatelské rozraní na ViewModel. ViewModel poskytuje veškerá data pro uživatelské rozhraní, nazývané View. Důležité je, že poskytuje svá data v takových datových strukturách, které vyvolávají události při jejich změně. To umožňuje uživatelskému rozhraní nová data automaticky zobrazit ihned jak se ve ViewModelu změní. ViewModel má dvě základní části. První je kolekce ObservableCollection, která hlásí přidání a odebrání prvku. Druhý je rozhraní INotifyPropertyChanged. Popisuje událost, která nastane, když se změní některá z vlastností ViewModelu. [7]

- Model - aplikací využívaná data
- View - uživatelské rozhraní
- ViewModel - spojuje data a uživatelské rozhraní a udržuje stav aplikace

4.3 Návrh řešení aplikace

Pro přehlednost aplikace, bude řešení rozděleno do základních pěti projektů. Dále jsou v programu využívány pomocné projekty sloužící jako knihovny, které jsou vlastnictvím společnosti Acam Solution. V následující části budou popsány základní využívané projekty.

4.3.1 Openness.Data

Openness.Data bude obsahovat třídy pro práci s databází. Databáze je využívána v aplikaci pro ukládání a nahrávání projektů. Jednotlivé projekty budou ukládány jako samostatné databáze. Pro databázi bude využit systém SQLite, který je volně dostupný s otevřenou licencí.

4.3.2 Openness.Models

Obsahuje návrh tříd aplikace představující entity v databázi

Třídou zařízení je DeviceModel. Ten obsahuje vlastnosti název, katalogové číslo, IP adresa, seznam bloků, seznam tabulek tagů a seznam UDT.

BlockModel bude třída bloků a obsahuje vlastnosti potřebné pro definici daného bloku, jako jsou název, typ bloku, číslo, skupina bloku a odkaz na vzor.

TagTableModel bude třída tabulek tagů, obsahuje vlastnosti jako název, odkaz na vzor tabulky, skupinu tabulky.

TagModel bude třída jednotlivých tagů.

UdtModel bude třída uživatelsky definovaných datových typů. Tato třída obsahuje vlastnosti název, skupina, datový typ, odkaz na vzor.

InstanceModel se bude využívat jako pomocný model pro vytvoření projektu v TIA portal. Každá instance obsahuje vazbu na Block (BlockModel), která udává jaký blok bude instance reprezentovat, ParentBlock (BlockModel), který odpovídá vnějšímu bloku z kterého je blok volán (například blok FC_MODE je volán z OB_MAIN) a ChildBlock (BlockModel), který odpovídá vnitřnímu bloku, který je z instance volán (opačný případ, v OB_MAIN je jako ChildBlock FC_MODE).

4.3.3 Openness.View

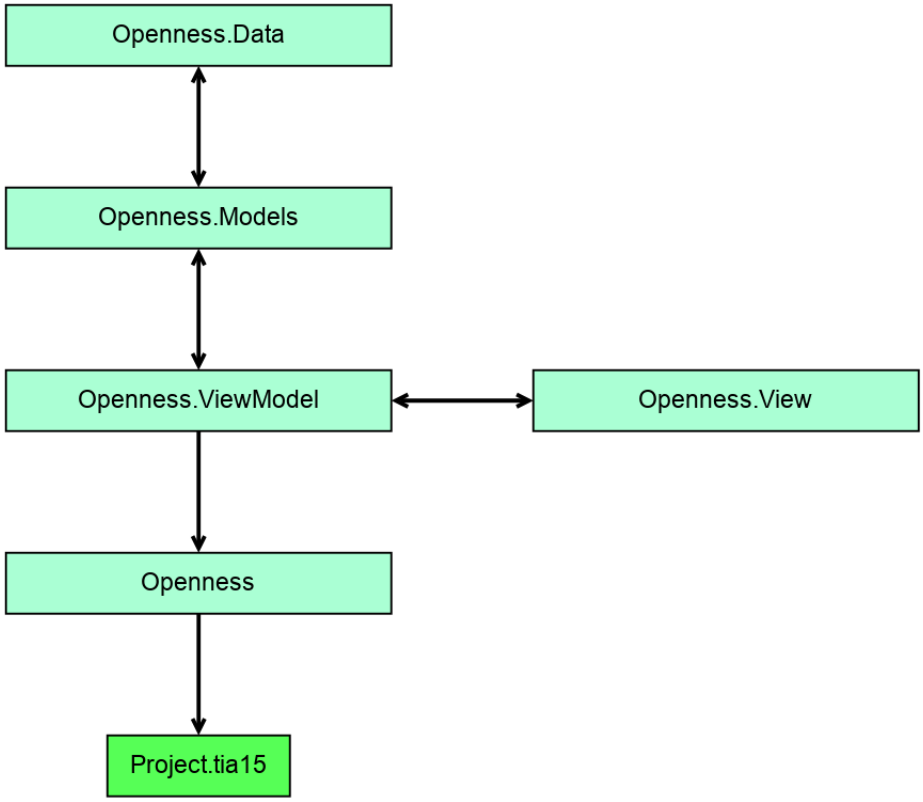
Openness.View bude WPF projekt, pomocí kterého lze vytvořit uživatelské rozhraní v jazyce XAML využívaném pro programování těchto aplikací. Vizuálně by aplikace měla být jednoduše ovladatelná a intuitivní. Díky vyžití návrhového vzoru MVVM je možné měnit typ uživatelského rozhraní a není nutné držet se WPF framworku.

4.3.4 Openness.ViewModel

Pomocí Openness.ViewModel se propojí datová část aplikace (Openness.Data, Openness.Models) a uživatelské rozhraní (Openness.View).

4.3.5 Openness

Projekt Openness vlastní třídu Openness, ve které jsou definovány potřebné metody pro generování projektu v TIA Portal V15.



Obr. 4.2: Schéma řešení aplikace

4.4 Návrh modelů

Modely, které bude aplikace využívat jsou vytvořeny v projektu Openness.Models. Každý model obsahuje vlastnosti korespondující s jeho využitím.

4.4.1 Model bloku

Základní model pro práci s organizačními, funkčními a datovými bloky a funkcemi v aplikačním prostředí. V tomto modelu je také definován typ výčtu typů bloků (BlockType), který obsahuje použitelné typy bloku.

Definované vlastnosti třídy:

GUID ID
string Name
string Description
BlockType BlockType
ObservableCollection<InstanceModel> Instances
ObservableCollection<InstanceModel> ChildInstances

- ID - univerzální unikátní identifikátor (Globally Unique Identifier) využívaný v databázi (primární, cizí klíč)
- Name - název bloku, název každého bloku by měl začínat dvojicí písmen definujících jeho typ (FC, OB, DB)
- Description - popis bloku, jako například funkční blok ovládání ventilů
- BlockType - typ bloku (FC, OB, DB, FB)
- Instances - instance které se odkazují na tento blok, například instance ventilů FB_YV11 a FB_YV12 jsou součástí dynamické kolekce bloku FB_VALVE
- ChildInstances - instance, které jsou v bloku volané, například FC_Stanice bude obsahovat instanci FB_YV11, což bude blok FB_VALVE.

4.4.2 Model tagu

Model definující PLC tag v aplikačním prostředí. V tomto modelu je také definován typ výčtu DataTypeName obsahující datové typy, které lze pro tag použít.

Definované vlastnosti třídy:

GUID ID
string Name
DataTypeName DataTypeName
string LogicalAddress
string Comment
GUID TagTableID
TagTableModel TagTable

- ID - univerzální unikátní identifikátor (Globally Unique Identifier) využívaný v databázi (primární, cizí klíč)
- Name - název tagu
- DataTypeName - datový typ tagu (Bool, Byte, Real, Word, Int)
- LogicalAddress - adresa tagu, každý tag má svou vlastní logickou adresu, první část adresy značí typ tagu (vstup, výstupm, merker), druhá část pozici v paměti PLC. Například vstupní tag typu bool může mít adresu %I10.0 a nachází se na 10 bytu a 0 bitu.
- Comment - popis, komentář k tagu
- TagTableID - cizí klíč definující vazbu na tabulku tagů
- TagTable - vazba na tabulku tagů v aplikaci

4.4.3 Model uživatelsky definovaného datového typu

Model uživatelsky definovaného datového typu (UDT) představuje vytvořené UDT v aplikačním prostředí

Definované vlastnosti třídy:

GUID ID
string Name
GUID DeviceID
DeviceModel Device

- ID - univerzální unikátní identifikátor (Globally Unique Identifier) využívaný v databázi (primární, cizí klíč)
- Name - název UDT
- DeviceID - cizí klíč definující vazbu na zařízení
- Device - vazba na vytvořené zařízení, které využívá danou UDT v aplikaci

4.4.4 Model Instancí

Instance jsou v aplikaci definované objekty, představující reálné bloky, které se budou v TIA projektu vytvářet. Každá instance má odkaz na blok, který představuje (OB_MAIN, FC_MODE, FB_VALVE,...), je tedy možné jeden definovaný blok vytvořit ve více variantách v TIA projektu. Nejčastěji je této vlastnosti využito u komponent (FB_VALVE, FB_MOTOR), kdy používáme stejný blok, ale vytváříme nové instance. Například FB_Stanice_1 bude obsahovat dva ventily. FB_YV11 a FB_YV12. Oba tyto funkční bloky jsou v programu instancemi (InstanceModel) s odkazem na blok FB_VALVE (BlockModel).

Definované vlastnosti třídy:

GUID ID
string Name
string Description
GUID ParentBlockID
BlockMode ParentBlock
GUID BlockID
BlockMode Block
GUID DeviceID
DeviceModel Device

- ID - univerzální unikátní identifikátor (Globally Unique Identifier) využívaný v databázi (primární, cizí klíč)
- Name - název instance (FB_Stanice_1, M11, YV12)
- Description - popis instance (odhrotovací stanice, chapadlo manipulátoru)
- ParentBlockID - cizí klíč definující vazbu na blok, ve kterém je instance volána
- ParentBlock - vazba na daný blok v programu
- BlockID - cizí klíč definující vazbu na blok, jenž je základem instance (FB_VALVE, FB_MOTOR)
- Block - vazba na daný blok v programu
- DeviceID - cizí klíč definující vazbu na zařízení
- Device - vazba na vytvořené zařízení, které využívá danou instanci v programu

4.4.5 Model tabulky tagů

Model definující tabulky tagů v aplikačním prostředí. V tomto modelu je také definován typ výčtu DataTypeName obsahující datové typy, které lze pro tag použít.

Definované vlastnosti třídy:

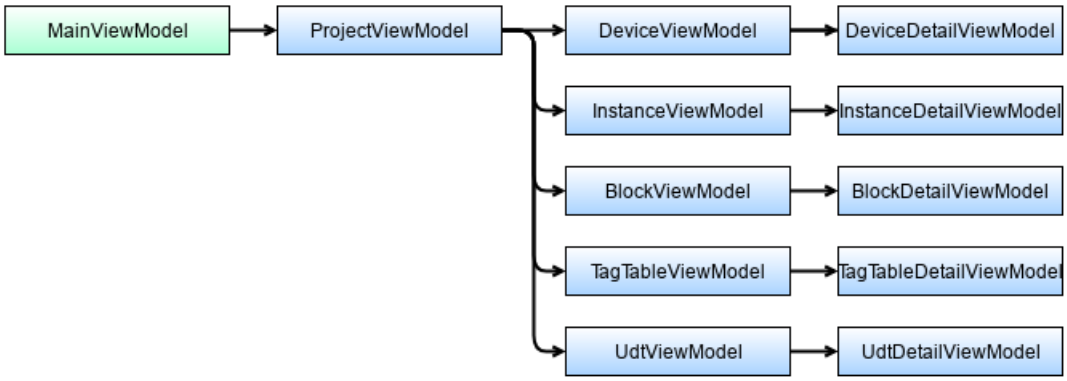
GUID ID
string Name
GUID DeviceID
DeviceModel Device
ObservableCollection<TagModel> Tags

- ID - univerzální unikátní identifikátor (Globally Unique Identifier) využívaný v databázi (primární, cizí klíč)
- Name - název tabulky tagů
- DeviceID - cizí klíč definující vazbu na zařízení
- Device - vazba na vytvořené zařízení, které využívá danou UDT v aplikaci
- Tags - dynamická kolekce tagů přiřazených k dané tagové tabulce

4.5 Návrh View-Modelu

ViewModely, které bude aplikace využívat pro propojení mezi datovou a vizualizační stránkou aplikace, jsou vytvořeny v projektu Openness.ViewModel. Hlavními View-Modely jsou MainViewModel a ProjectViewModel, které spravují základní provázání View s ViewModely. Dále jsou ke každému vytvořenému modelu přiřazeny dva typy ViewModelu, a to ViewModel a DetailViewModel. ViewModel bude využíván pro práci s celou sadou uložených objektů daného modelu (BlockMode, DeviceModel, atd.). DetailViewModel je vlastností ViewModelu (ItemDetail) a bude spravovat jednotlivé objekty vybrané ze sady ViewModelu.

Příkladem můžou být nadefinovány bloky OB_MAIN, FC_CONTROL. Formulář zobrazující tyto bloky ve View programu bude navázán na BlockViewModel, ovšem formuláře spravující jednotlivé prvky, například OB_MAIN, už budou navázány na BlockDetailViewModel. Tímto způsobem se zpřehlední práce s jednotlivými objekty a jejich navázání na view programu.



Obr. 4.3: Návrh ViewModelu

4.5.1 MainViewModel

Jedná se o hlavní ViewModel, který je navázán na hlavní okno aplikace a spravuje vytváření a otevírání projektů.

Vlastnosti:

string ConnectionString

Takzvaný připojovací řetězec, zprostředkovává připojení databáze. V mém případě se ConnectionString bude rovnat názvu vytvořeného projektu.

ObservableCollection<ProjectViewModel> Projects

Dynamická kolekce projektů. Je navázána na veškeré aktuálně otevřené projekty v aplikaci.

Příkazy:

ICommand CreateProjectCommand;

ICommand OpenProjectCommand;

Metody:

public void CreateProject();

Metoda pro vytvoření nového projektu v aplikaci. Využívá příkaz CreateProjectCommand, který je navázán na tlačítko v záhlaví aplikace. Po spuštění této metody se otevře okno pro zadání názvu projektu a místa uložení. Po potvrzení je vytvořen nový ProjectViewModel a přidán do kolekce Projects.

public void OpenProject();

Metoda pro otevření projektu v aplikaci. Využívá příkaz OpenProjectCommand, který je navázán na tlačítko v záhlaví aplikace. Po spuštění této metody se otevře okno pro výběr projektu na disku. Po potvrzení je daný projekt propagován do aplikace a přidán do kolekce Projects.

public void Initialize(string path, ProjectViewModel project);

Pomocí InitializeBlocks, InitializeTagTables a InitializeUdts lze při vytvoření nového projektu inicializovat seznamy předem definovaných objektů. Parametr path obsahuje cestu k výchozím xml souborům, které budou v projektu navázány.

4.5.2 ProjectViewModel

Základní okno aplikace MainWindow.xaml je navázáno na MainViewModel, toto okno má tzv. tab control. Tento tab control je navázán na dynamickou kolekci ProjectViewModelů. Tato třída vytváří zbylé ViewModely ve svém konstruktoru.

Vlastnosti:

string ConnectionString

Takzvaný připojovací řetězec, zprostředkovává připojení databáze. Je definován při konstrukci objektu.

string ProjectPath

Cesta k projektu na disku. Je definován při konstrukci objektu.

ObservableCollection<ProjectViewModel> ActualProjects

Aktuálně otevřené projekty v aplikaci.

BlockViewModel BlockVM

Základní ViewModel pro bloky. Je inicializován při vytvoření objektu ProjectViewModel.

InstanceViewModel InstanceVM

Základní ViewModel pro instance. Je inicializován při vytvoření objektu ProjectViewModel.

DeviceViewModel DeviceVM

Základní ViewModel pro zařízení. Je inicializován při vytvoření objektu ProjectViewModel.

UdtViewModel UdtVM

Základní ViewModel pro UDT. Je inicializován při vytvoření objektu ProjectViewModel.

TagTableViewModel TagTablesVM

Základní ViewModel pro tabulky tagů. Je inicializován při vytvoření objektu ProjectViewModel.

Příkazy:

ICommand CreateProjectCommand;

ICommand CloseProjectCommand;

Metody:

public void CloseProject();

Zavření aktuálně používaného projektu. Metoda je navázána na příkaz CloseProjectCommand.

public void GetActualProjects(ObservableCollection<ProjectViewModel> actualProjects);

Zjištění aktuálně otevřených projektů.

public void CreateProject();

Pomocí této metody se vytváří projekt v TIA portal. Metoda je navázána na okno "Vytvořit projekt v TIA Portal", v tomto okně jsou zaškrtávací políčka s různými možnostmi. Na základě výběru pak metoda provede vytvoření projektu. Nejprve se aplikace připojí k instanci TIA portálu, jakmile je toto spojení provedeno, vytvoří se projekt dle parametrů nastavených uživatelem. Následně se do projektu přidají veškerá zařízení. Jakmile jsou zařízení úspěšně přidána, nahrají se příslušné tabulky tagů, UDT, a bloky. Při nahrávání bloků se nejprve importují komponenty, poté stanice a následně zbylé bloky. Po přidání všech bloků je provedena pro každé zařízení kompilace a projekt je uložen.

4.5.3 Obecné metody základních ViewModelů

Tyto metody jsou využívány všemi vytvořenými základními ViewModely (BlockViewModel, DeviceViewModel, UdtViewModel, TagTableViewModel).

public void Refresh();
Zjištění aktuálního stavu dat navázaných na daný ViewModel a definovaných v databázi.

public void InsertNewItem();
Vytvoření nového záznamu daného modelu (BlockModel, DeviceModel, atd.). Tato metoda přidává novou instanci DetailViewModel do kolekce dat ItemDetail.

public void CopyItem();
Vytvoření nového záznamu kopírováním již vytvořeného DetailViewModelu.

public void ShowItemDetail();
Zobrazení ItemDetail daného ViewModelu.

4.5.4 Speciální metody základních ViewModelů

Tyto metody jsou využívány pouze některými ViewModely.

BlockViewModel
ICommand AddMultipleBlocksCommand;
public void AddMultipleBlocks();

Tato metoda je využívána pro přidání/vytvoření většího počtu bloků najednou. Je navázána na příkaz AddMultipleBlocksCommand. Ten pokud je spuštěn z formuláře, objeví se okno pro výběr většího počtu souborů. Tyto soubory musí být definované xml bloky pro PLC. Metoda následně vyhodnotí jednotlivé soubory a na základě jejich dat nastaví název, popis a veškeré další potřebné vlastnosti modelu bloku.

TagTableViewModel
ICommand AddMultipleTagTablesCommand;
public void AddMultipleTagTables();

Metoda je využívána pro přidání/vytvoření většího počtu tabulek tagů najednou. Je navázána na příkaz AddMultipleTagTablesCommand. Ten pokud je spuštěn z formuláře, objeví se okno pro výběr většího počtu souborů. Tyto soubory musí být definované xml tabulky tagů pro PLC. Metoda následně vyhodnotí jednotlivé soubory a na základě jejich dat nastaví název, popis a veškeré další potřebné vlastnosti modelu tabulek tagů.

UdtViewModel
ICommand AddMultipleUdtCommand;
public void AddMultipleUdt();

Metoda je využívána pro přidání/vytvoření většího počtu uživatelsky definovaných datových typů najednou. Je navázána na příkaz AddMultipleUdtCommand. Ten pokud je spuštěn z formuláře, objeví se okno pro výběr většího počtu souborů. Tyto soubory musí být definované xml UDT pro PLC. Metoda následně vyhodnotí jednotlivé soubory a na základě jejich dat nastaví název, popis a veškeré další potřebné vlastnosti modelu UDT.

4.5.5 Obecné metody základních DetailViewModelů

Veškeré DetailViewModely mají definovány následující příkazy a metody.

Příkazy:
ICommand RefreshCommand;
ICommand SaveCommand;
ICommand RemoveCommand;
ICommand CancelCommand;

Metody:
public void Refresh();
Zjištění aktuálního stavu dat navázaných na daný DetailViewModel a definovaných v databázi.

public void Save();
Uložení aktuálních hodnot do databáze.

public void SaveFile();
Metoda, využívající příkaz SaveCommand, otevře okno prohlížeče souborů a čeká na výběr souboru. Po vybrání souboru se vytvoří vazba na objekt v programu a zároveň kopie ve složce UmisteniProjektu/Data/GUID_daného_objektu. Využívá se pro navázání definovaných xml souborů (bloky, udt, tabulky tagů) s daty aplikace.

public void CanSaveFile();
Metoda zjišťující povolení vytvoření vazby na soubor.

public void RemoveFile();
Zrušení vazby souboru na daný objekt v programu.

public void CanRemoveFilef();
Metoda zjišťující povolení smazání souboru.

public void OpenFile();
Pomocí této metody lze zobrazit v prohlížeči aktuální xml soubor, který je navázán na daný objekt.

public void CanOpenFile();
Metoda zjišťující povolení otevření souboru.

4.5.6 Speciální metody základních DetailViewModelů

Tyto metody jsou specifické pro jednotlivé DetailViewModely.

BlockDetailViewModel

ICommand AddInstanceCommand;
ICommand RemoveInstanceCommand;
public void AddInstance();
Metoda po aktivaci příkazu AddInstanceCommand, který je navázán na tlačítko (+) v okně BlocksControl.xaml, otevře dialogové okno s výběrem vytvořených bloků. Po zvolení příslušného bloku je uživatel dotázán na název a popis instance. Metoda následně pomocí těchto informací vytvoří příslušnou instanci.

public void RemoveInstance();
Po aktivaci příkazu RemoveInstanceCommand, který je navázán na tlačítko (-) v okně BlocksControl.xaml, je příslušná instance vymazána, včetně veškerých vazeb v projektu.

public void CanRemoveInstance();
Pomocí této metody je zjištěno, zda je možné příslušnou instanci odstranit. Metoda kontroluje, zda vybraná instance ke smazání není null a zároveň blok tuto instanci obsahuje.

public void AddToXmlFile(BlockModel childBlock, InstanceModel instance);
Vytvoření instance, z pohledu projektu v TIA portal, znamená volání daného bloku v bloku jiném, například funkční blok ventilu ve funkčním bloku stanice. Je tedy potřeba daný blok změnit. Tyto změny se provádí v xml souboru, kterým je blok definován.

Metoda si nejprve na základě ID bloku, do kterého je instance přidávána, najde xml soubor a následně podle typu bloku (komponenta, stanice, OB_Main, atd.) jej pozmění. Pro instance volané v OB_Main jako jsou stanice a řídící funkce stačí pouze přidání takzvaných CompileUnits, což je část kódu v xml souboru, která definuje příslušný network (síť - část kódu v logice LAD). U instancí typu komponenta je potřeba přidat také vstupy, výstupy, příkazy a instanční datový blok náležící danému funkčnímu bloku, kterým je komponenta definována.

public void RemoveFromXmlFile(BlockModel childBlock, InstanceModel instance);

Tak jako je třeba přidávat instance k jednotlivým blokům je důležité umět tyto instance odstranit. Metoda RemoveFromXmlFile odstraní příslušnou instanci z xml souboru bloku, ve kterém byla definována. Pokud se jedná o komponentu, jsou také odstraněny instanční datové bloky, příkazy, vstupy a výstupy pro danou komponentu.

TagTableDetailViewModel

ICommand AddTagCommand;
ICommand RemoveTagCommand;
public void AddTag();

Pomocí metody AddTag, která je využívána při spuštění příkazu AddTagCommand, lze přidat nový tag do tabulky tagů. Metoda vyvolá okno, ve kterém uživatel vybere datový typ tagu a zadá název s adresu tagu. Tímto se vytvoří vazba mezi tagem a tabulkou. Zároveň je upraven xml soubor tagové tabulky tak, aby následná tabulka vytvořená v TIA portalu obsahovala takto přidané tagy.

public void RemoveTag();

Po aktivaci příkazu RemoveTagCommand, se odstraní vazba na tabulku tagů, včetně části kódu v souboru xml odkazujícího na tuto tabulku.

DeviceDetailViewModel

ICommand AddInstanceCommand;
ICommand RemoveInstanceCommand;
ICommand AddTagTableCommand;
ICommand RemoveTagTableCommand;
ICommand AddUdtCommand;
ICommand RemoveUdtCommand;
public void AddInstance();

Metoda AddInstance objektu DeviceDetailViewModel je volána, pokud chce uživatel přidat organizační nebo datový blok do zařízení. Metoda vyvolá okno, do kterého uživatel zadá název instance a odkaz na žádaný blok. Tímto způsobem se vytváří vazba mezi zařízením a bloky.

public void RemoveInstance();

Pomocí RemoveInstance je odstraněna vazba mezi instancí a zařízením.

public void AddTagTable();

Vytváří vazbu mezi definovanými tabulkami tagů a zařízením. Vyvolá okno, z kterého si uživatel vybere příslušnou tabulku tagů a potvrdí. Tím se daná vazba vytvoří.

public void RemoveTagTable();

RemoveTagTable odstraní vazbu mezi vybranou tabulkou tagů a zařízením.

public void AddUdt();

Vytváří vazbu mezi definovaným UDT a zařízením. Podobně jako u AddTagTable, vyvolá okno, z kterého si uživatel vybere příslušné UDT a potvrdí.

public void RemoveUdt();

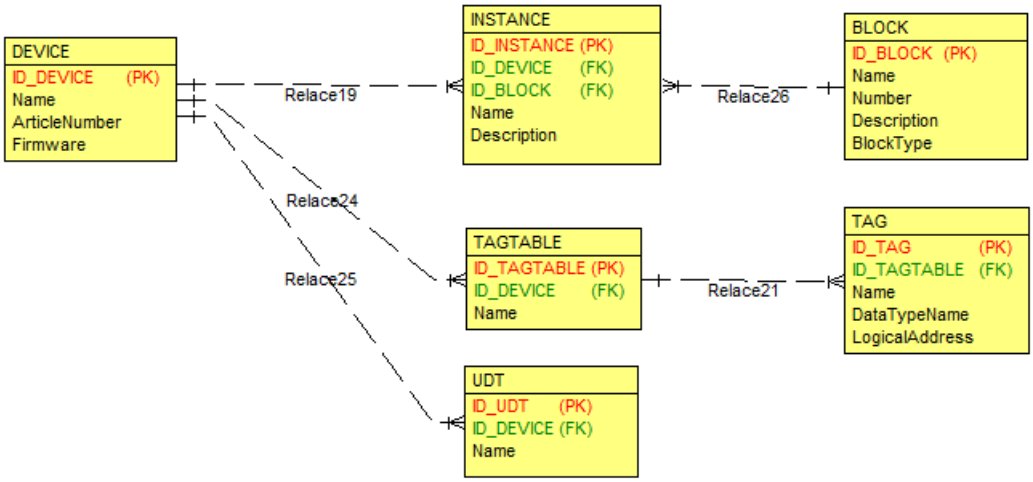
Odstraní vazbu mezi vybraným UDT a zařízením.

4.6 Návrh databáze

Databáze aplikace je vytvořena na základě modelů z Openness.Models. Každá entita je definována dle příslušné třídy. Atributy následně odpovídají jednotlivým vlastnostem třídy.

Pro práci s databází byl využit .NET Entity Framework, pomocí kterého lze jednoduše mapovat objekty definované v softwaru s tabulkami a sloupci relační databáze. Aplikace inicializuje databázi pomocí objektu OpennessDataDbContext vytvořeného v projektu AVIC.Openness.Data/Base. Pomocí metody OnModelCreating jsou vytvořené příslušné entity a vazby.

Na následujícím obrázku můžeme schéma vytvořené databáze.



Obr. 4.4: Schéma databáze projektu

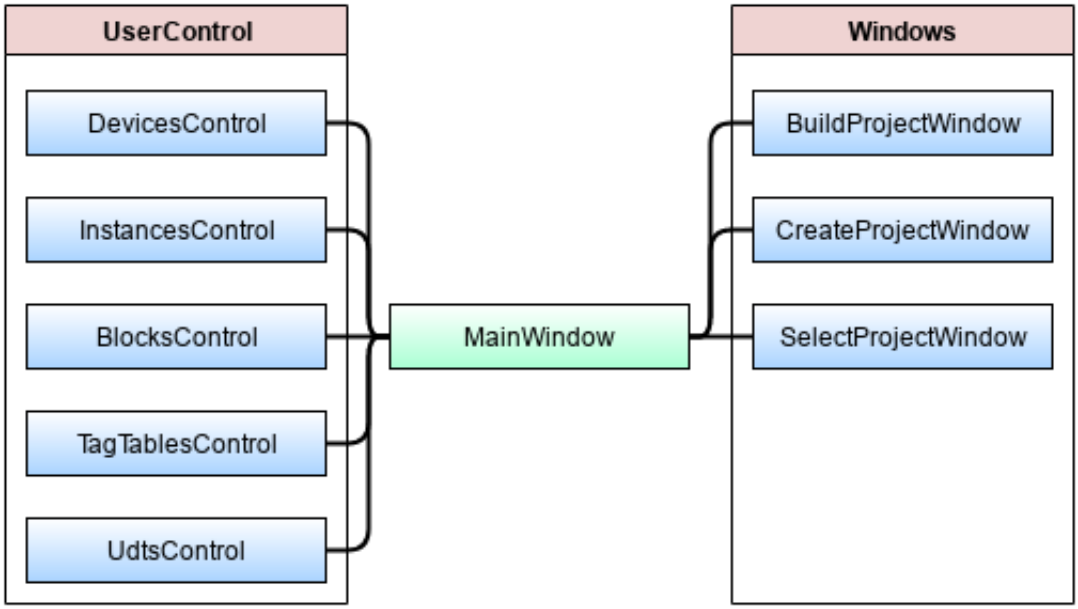
4.7 Návrh grafického rozhraní

Grafické rozhraní vytvořené pomocí Windows Presentation Foundation v projektu AVIC.Openness.WPF bude rozděleno na tři části.

První část je hlavní okno aplikace definované v MainWindow.xaml. Toto okno bude neustále otevřené a veškeré akce, které uživatel provede, se na něm budou promítat. MainWindow je navázáno na MainViewModel.

Druhá část WPF projektu se nazývá UserControl (uživatelské ovládání). Tyto okna zobrazují jednotlivé objekty definované v Openness.Models jako jsou bloky, zařízení, tabulky tagů a podobně. UserControl okna jsou navázány na základní ViewModely příslušných objektů (BlockViewModel) a budou dostupná na MainViewModelu pomocí tabControlu.

Třetí část projektu jsou tzv. Windows, tedy okna, která budou mít jeden účel. Například okno pro vytvoření nového projektu nebo otevření projektu. Pro tyto xaml soubory budou definovány také příslušné ViewModely ve složce ViewModels/-Dialogs.



Obr. 4.5: Schéma grafického rozhraní

5 Vytvořená aplikace

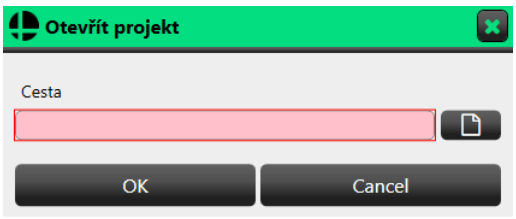
Aplikace pro svou funkčnost vyžaduje nainstalovaný TIA portal V15 s platnou licencí a uživatelský účet počítače musí být přidán do skupiny Siemens TIA Openness. Aplikaci lze spustit pomocí exe souboru AVIC.Openness.WPF.exe.

5.1 Vytvoření projektu

Uživateli se při spuštění aplikace zobrazí hlavní okno s možností vytvořit a nebo založit nový projekt.

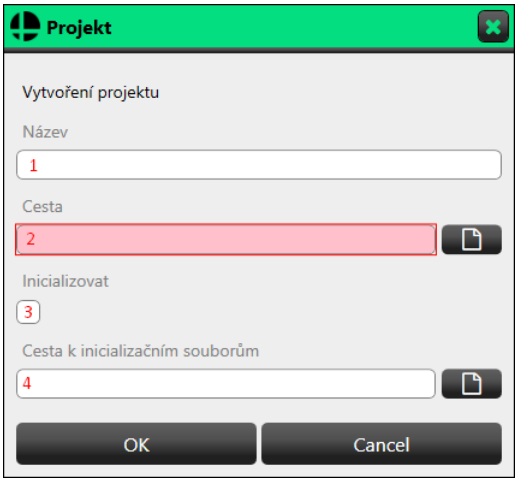
Obr. 5.1: Hlavní okno aplikace

Při zvolení Otevřít. Je zobrazeno textové okno. Uživatel je požádán o zadání cesty k projektu, který chce otevřít.



Obr. 5.2: Otevřít projekt

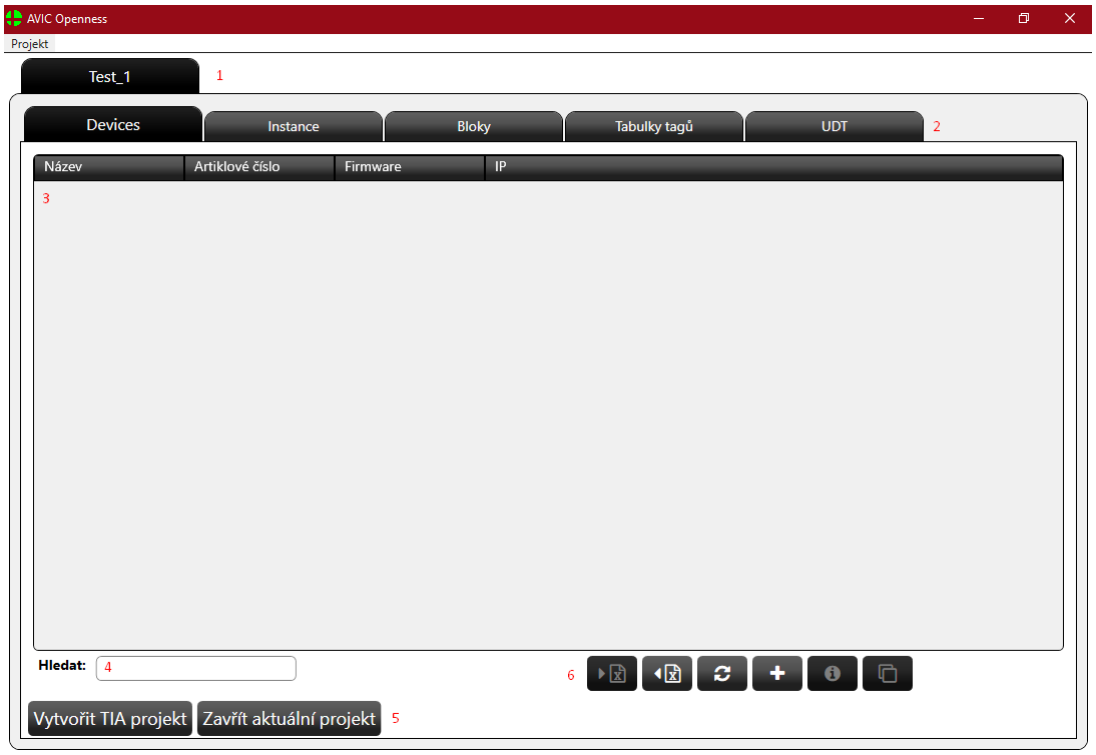
Pokud uživatel zvolí vytvořit projekt, zobrazí se následující okno.



Obr. 5.3: Vytvořit projekt

1. Název projektu
2. Cesta kde má být projekt vytvořen
3. Pokud uživatel zaškrtne, projekt bude inicializován výchozími bloky, tagy a UDT.
4. Cesta k inicializačním souborům

Po vytvoření nebo otevření projektu je uživateli zobrazena projektová obrazovka s potřebnými záložkami.



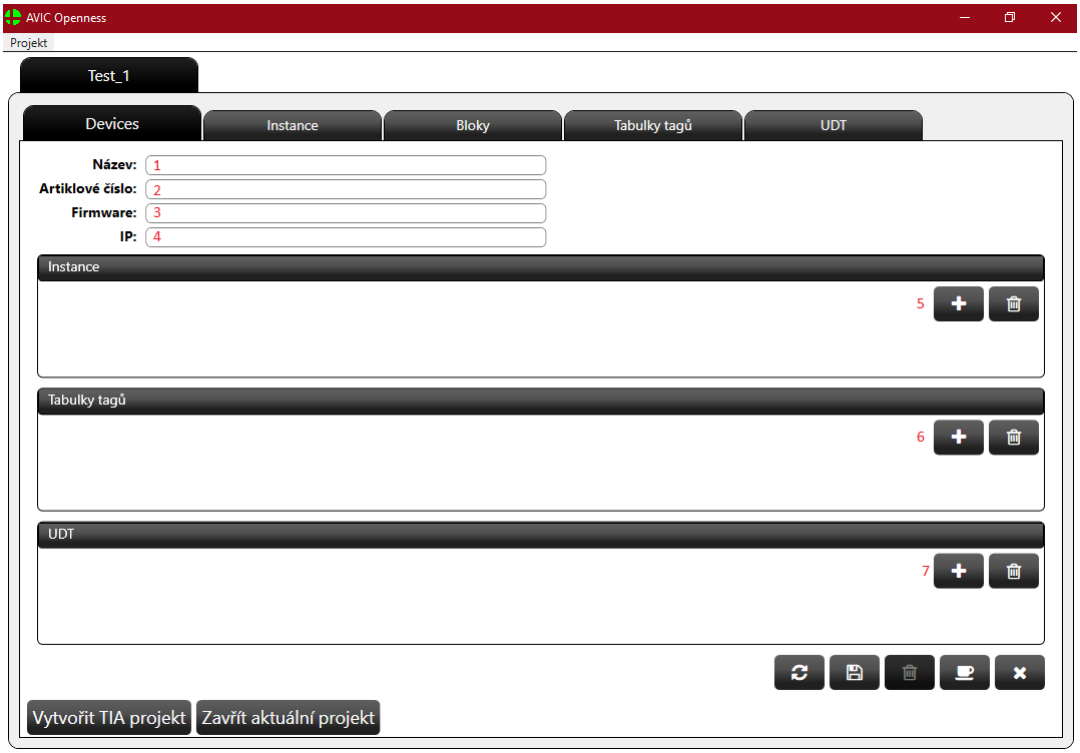
Obr. 5.4: Projektová obrazovka

1. Záložky aktuálně otevřených projektů
2. Záložky zařízení, instancí, bloků, tabulek tagů a udt projektu
3. Zobrazení vytvořených prvků
4. Vyhledávání v aktuální kartě
5. Tlačítko pro vytvoření projektu v TIA portal a pro zavření aktuálního projektu
6. Tlačítka pro práci s jednotlivými prvky karty

5.2 Vytvoření prvků a vazeb v projektu

5.2.1 Přidání zařízení

Zařízení lze přidat v záložce Devices, kliknutím na tlačítko (+). Tím se zobrazí okno detailního pohledu, kde lze vyplnit potřebné údaje o zařízení.

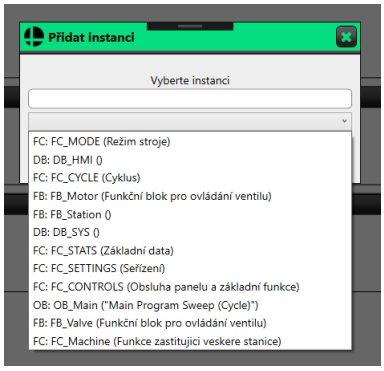


Obr. 5.5: Přidání zařízení

1. Název zařízení
2. Artiklové číslo, jedná se o jedinečný identifikátor hardwaru od společnosti siemens.
3. Firmware zařízení
4. IP adresa zařízení
5. Tlačítka pro přidání a odebrání instance zařízení
6. Tlačítka pro přidání a odebrání tabulky tagů zařízení
7. Tlačítka pro přidání a odebrání udt zařízení

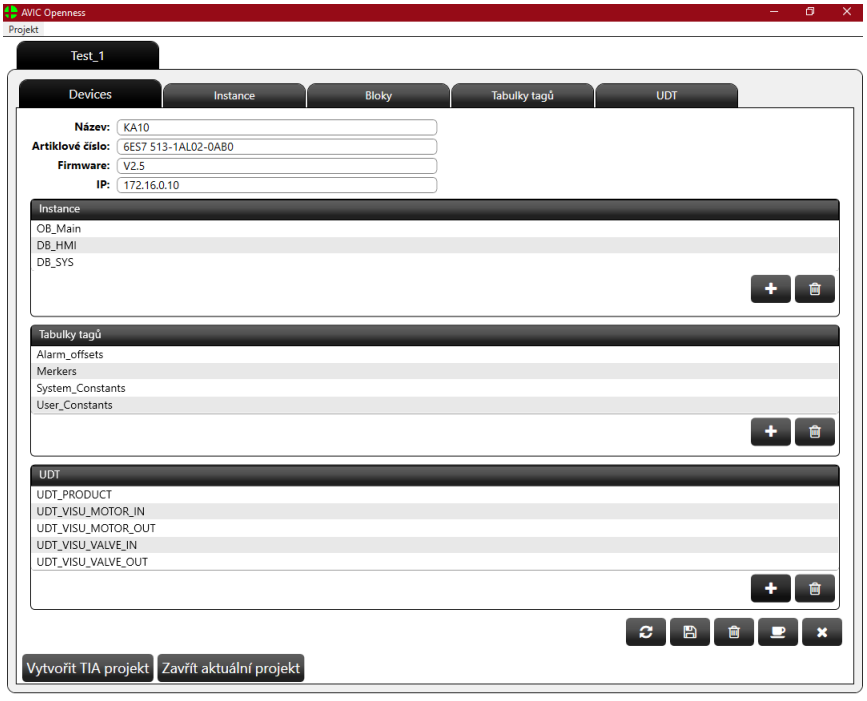
5.2.2 Vytvoření základních vazeb na zařízení

Jakmile je definováno zařízení, je potřeba vytvořit vazby mezi vytvořenými instancemi, tabulkami tagů a udt a naším zařízením. Podle navržené struktury projektu v TIA portal, víme, že přímo na zařízení jsou navázány pouze organizační a datové bloky. Pomocí tlačítka AddInstance (+) vytvoříme instanci a přímou vazbu na zařízení z DB_HMI, DB_SYS a OB_Main.



Obr. 5.6: Vytvoření instance pomocí zařízení

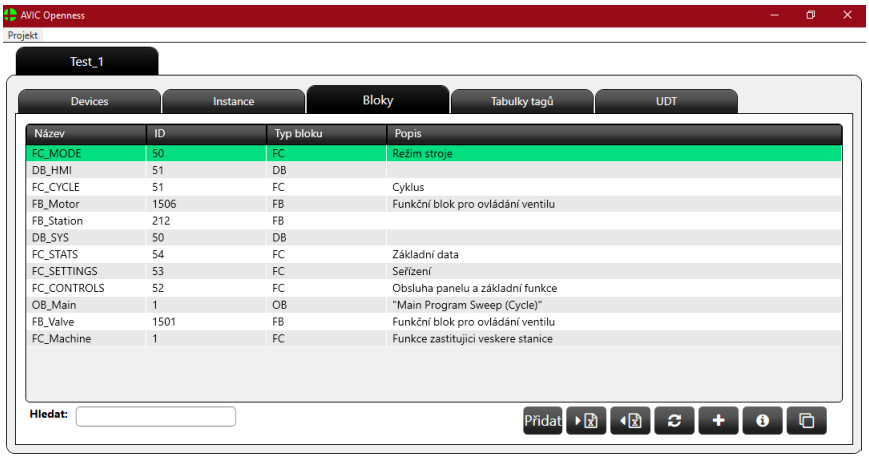
Stejným způsobem vytvoříme vazby na požadované tabulky tagů a udt.



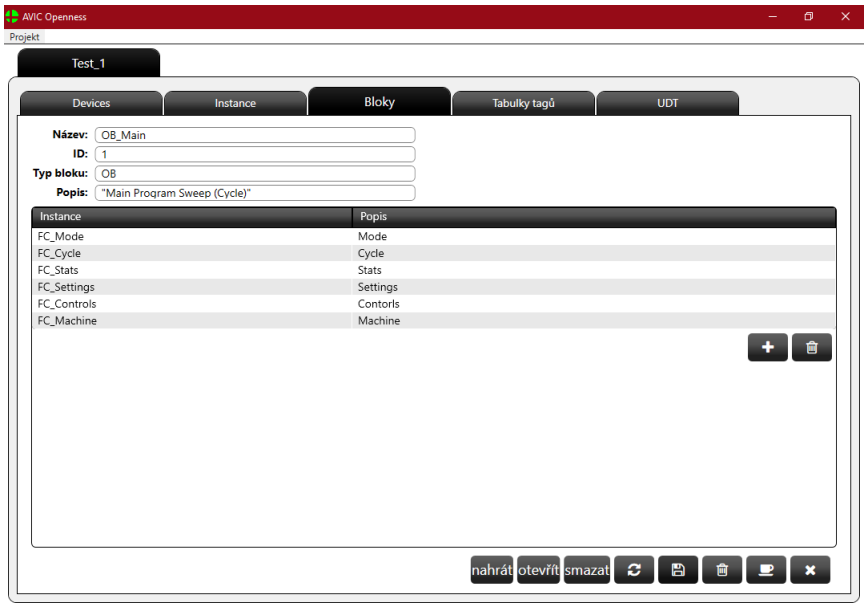
Obr. 5.7: Zařízení s nastavenými vazbami

5.2.3 Vytvoření vazeb v OB_Main

Podle předem navržené struktury víme, že v OB_Main jsou volány veškeré funkce a funkční bloky. Proto je potřeba nastavit vazby reprezentující toto volání. V kartě Blocks se po dvojkliku na OB_Main zobrazí detail tohoto bloku. Zde se nastaví instance tohoto bloku (FC_Mode, FC_Settings, atd.). Na prvním obrázku můžeme vidět výchozí bloky, inicializované při založení projektu. Na druhém obrázku následně detail bloku OB_Main s vazbami na jednotlivé instance.



Obr. 5.8: Výchozí bloky



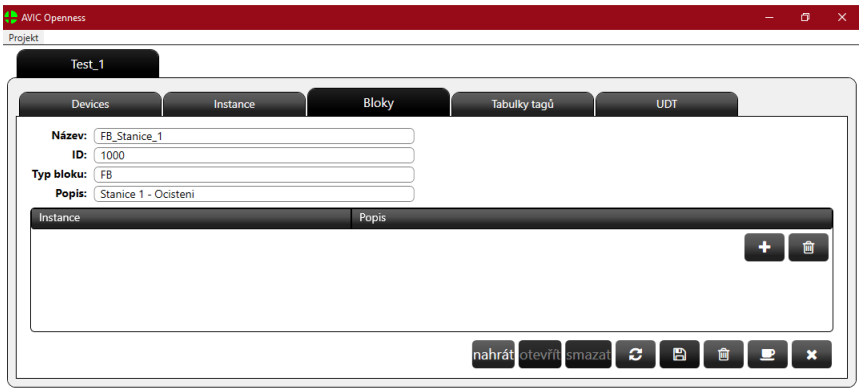
Obr. 5.9: Inicializován OB_Main

5.2.4 Vytvoření a navázání stanic

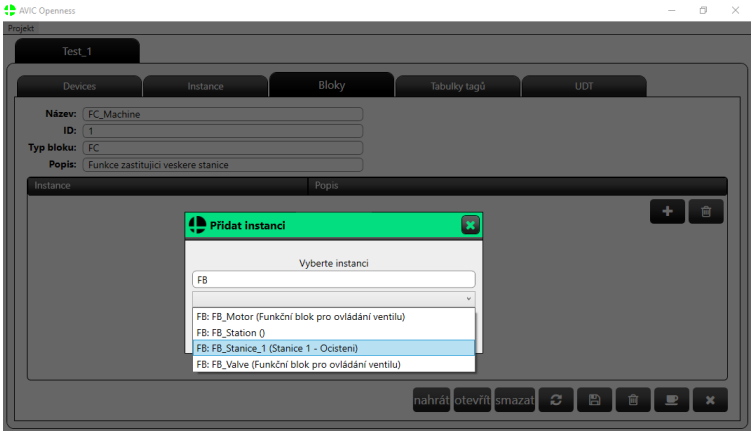
Z navrženého schématu programu v PLC víme, že všechny stanice jsou volány z FC_Machine. V předchozím kroku jsme vytvořili vazbu FC_Machine na OB_Main. Nyní je potřeba FC_Machine naplnit stanicemi.

Jako první vytvoříme blok představující naší stanici. Vyplníme požadované parametry bloku a uložíme. Jakmile jsme blok uložili, je potřeba jej navázat na xml soubor s výchozí podobou stanice. Docílíme toho stiskem tlačítka nahrát a vybráním daného souboru. Navázaný soubor lze otevřít, nebo smazat pomocí stejně pojmenovaných tlačítek.

Následně je potřeba navázat tuto stanici na FC_Machine (FC_Stroj). Otevřeme si tedy blok FC_Machine a pomocí tlačitka (+) vytvoříme novou instanci FB_Stanice_1 odkazující na stejně pojmenovaný blok.

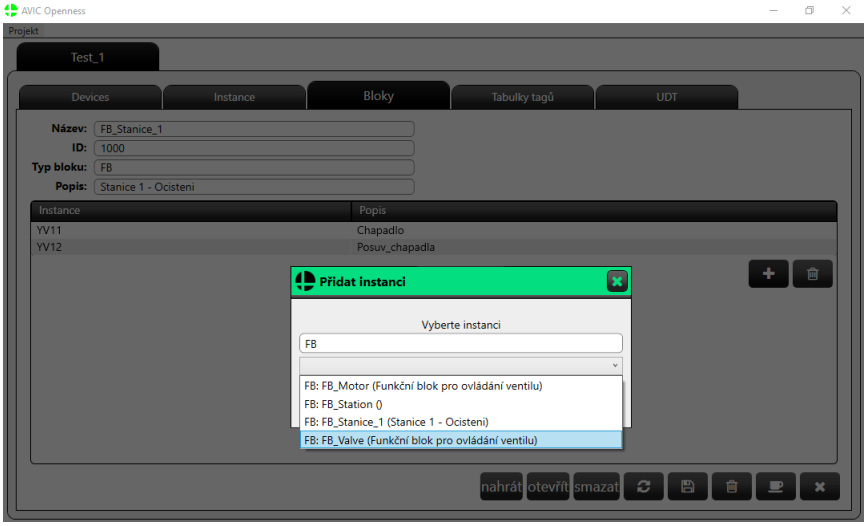


Obr. 5.10: Vytvoření stanice



Obr. 5.11: Navázání stanice na FC_Machine

Nakonec do stanice přidáme potřebné ventily. Po otevření detailu FB_Stanice_1 můžeme tlačítkem (+) přidat novou instanci. Vybereme blok FB_Valve, funkční blok pro ovládání ventilů, a vyplníme název a popis instance (ventilu).



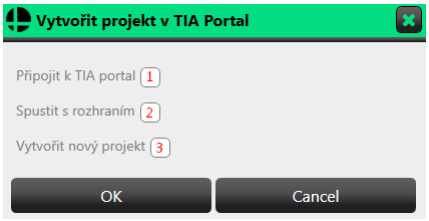
Obr. 5.12: Přidání komponenty do stanice

5.3 Projekt v TIA portal

Pokud jsou vytvořeny všechny potřebné vazby mezi jednotlivými prvky projektu, můžeme vytvořit projekt v TIA portal V15.

5.3.1 Vytvoření projektu v TIA portal

Pomocí tlačítka v kartě Devices, Vytvořit TIA projekt zobrazíme okno s parametry pro vytvoření projektu.



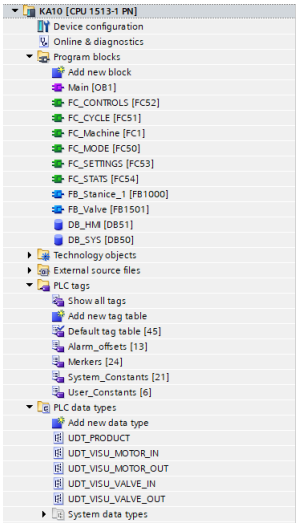
Obr. 5.13: Vytvoření projektu v TIA portal

1. Po zaškrtnutí, není při vytváření projektu otevřena nová instance TIA portal aplikace a nevytvoří se nový projekt (celý projekt bude vytvořen v aktuálně otevřeném projektu v TIA portal)
2. Výběr zda uživatel požaduje grafické rozhraní TIA portálu při vytváření projektu
3. Po zaškrtnutí se vytvoří nový projekt

Odsouhlasením tlačítkem OK je otevřena aplikace TIA portal V15 a požádá o povolení. Aby mohl program pokračovat je potřeba tuto žádost potvrdit. Následně je vytvořen projekt, naimportovány veškeré objekty a provedena kompilace zařízení.

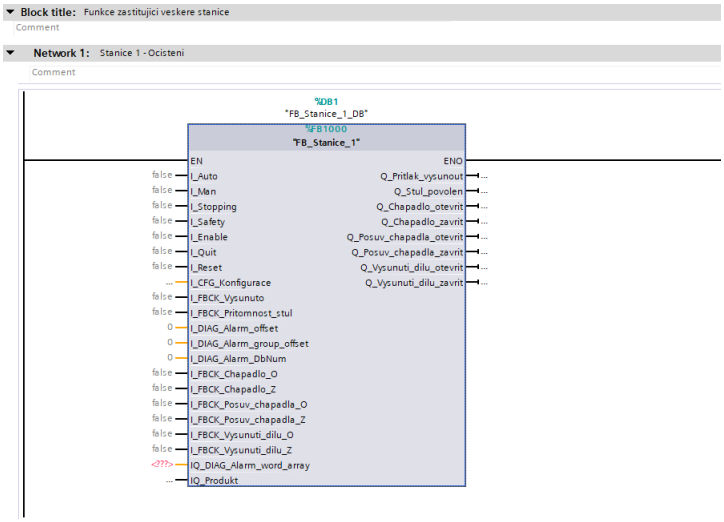
5.3.2 Vytvořený projekt

Kompilací zařízení končí proces vytváření projektu v TIA portal a lze s ním manipulovat. Projekt je vytvořen podle vazeb bloků a instancí v aplikaci. Importují se navázané UDT, tabulky tagů a využívané bloky.



Obr. 5.14: Vytvořený projekt v TIA portal pomocí openness

Podle návrhu, jsou stanice volány v FC_Machine. V projektu jsme měli nakonfigurovanou stanici FB_Stanice_1. Na obrázku můžeme vidět její reprezentaci v FC_Machine. Následující obrázek ukazuje nakonfigurované I_FBCK vstupy a výstupy Q pro jednotlivé komponenty.

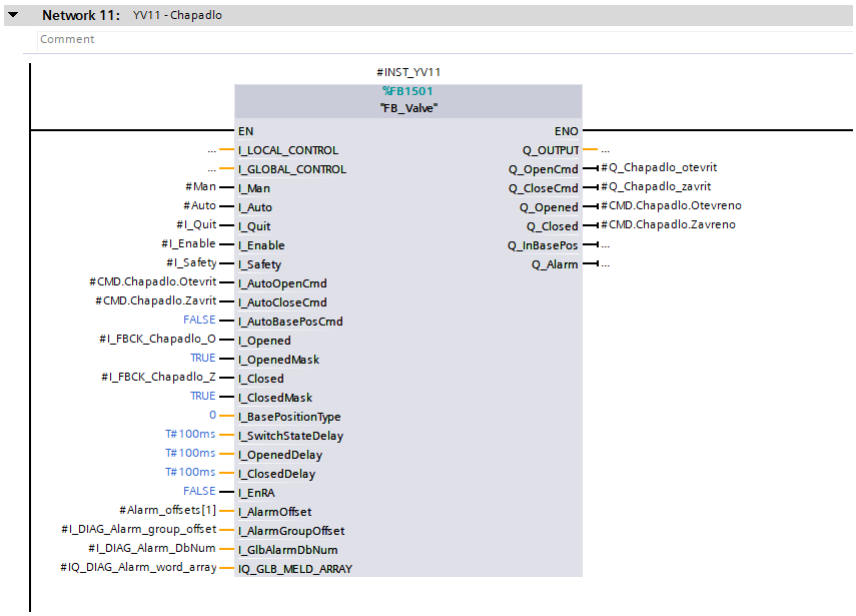


Obr. 5.15: Funkce FC_Machine s vytvořenou stanicí

Stanice FB_Stanice_1 má nakonfigurovány tři ventily a to YV11, YV12 a YV13. Jak můžeme vidět na obrázku, ovládání těchto ventilů bylo vytvořeno v části HW KOMPONENTY podle návrhu struktury projektu.

Obr. 5.16: Komponenty vytvořené stanice

Na následujícím obrázku se nachází jeden z komponentů ve stanici. Můžeme si všimnout, že má nakonfigurovaný instanční datový blok (INST_YV11). Dále jsou nakonfigurovány příkazové proměnné (CMD), vstupy zpětných vazeb (I_FBCK) a výstupy ovládající reálné ventily (Q). Navázány jsou také veškeré řídící proměnné stanice (Man, Auto, atd.). Tímto způsobem jsou nakonfigurovány veškeré komponenty ve stanici (YV11, YV12, YV13).



Obr. 5.17: Vytvořená komponenta ve stanici

Výše popsaný projekt naleznete v příloze A. Veškeré soubory, které obsahuje složka Priloha_projekt byly vytvořeny pomocí této aplikace.

Závěr

Práce obsahuje přehled dostupných nástrojů pro automatické generování kódu pro PLC Siemens. Komerčně dostupné nástroje pro automatické generování kódu PLC jsou vhodné pro firmy s dostatečným počtem zakázek, aby byla zajištěna finanční návratnost licence. Ve většině případů bylo velmi těžké zjistit veškeré informace o daném softwaru, kdy prodejci nepodávají údaje o přesné funkčnosti jejich produktu.

Byl vypracován přehled možností knihovny Siemens TIA Openness. Knihovna byla využita pro vytvoření vlastní aplikace na automatické generování kódu v TIA Portal.

Pomocí programu Siemens TIA Portal V15 byla navržena základní struktura projektu, včetně veškerých potřebných bloků, tabulek tagů a uživatelsky definovaných datových typů, pro implementaci generování kódu pomocí TIA Openness.

V jazyce C# byla vytvořena uživatelská aplikace pomocí návrhového vzoru Model-View-ViewModel. Aplikace byla otestována a umožňuje vytvoření projektu v TIA portal v15. Uživatel má možnost v aplikaci vytvářet projekty obsahující zařízení, bloky, tabulky tagů, tagy a UDT. Pro každé zařízení v projektu lze definovat vazby na jednotlivé bloky, tabulky tagů a UDT, a to na základě navržené struktury projektu v TIA portal.

Literatura

[1] ANDERSSON, Mikael a Erik HELANDER. Automatic generation of PLC programs using Automation Designer. Göteborg, Sweden, 2010. Master thesis. Chalmers University of Technology.

[2] Neue Wege zum Digitalen Zwilling durch mechatronisches AnlagenEngineering [online]. Germany: Siemens, 2018 [cit. 2019-10-28]. Dostupné z: https://drive.google.com/file/d/1XZUEFNEC1BdIwF9YQQ-IEsaJgu6a_Ayy/view?usp=sharing

[3] Simulink PLC Coder [online]. United States: The MathWorks [cit. 2019-10-28]. Dostupné z: https://www.mathworks.com/products/simulink-plc-coder.html

[4] AutoGen. RIVOPS [online]. Norsko: Remote Intuitive Visual Operations, 2018 [cit. 2019-11-03]. Dostupné z: http://www.rivops.com/autogen

[5] Openness: Automating creation of projects: System Manual. 10/2018. Germany: Siemens, 2018.

[6] Úvod do WPF (Windows Presentation Foundation) [online]. 2015 [cit. 2020-01-03]. Dostupné z: https://www.itnetwork.cz/csharp/formulare/wpf/c-sharp-tutorial-wpf-uvod-a-prvni-formularova-aplikace/

[7] DAJBYCH, Václav. MVVM: model-view-viewmodel [online]. 21.04.2009 [cit. 2020-01-03]. Dostupné z: https://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[8] SIEMENS. SIMATIC Automating projects with scripts: System Manual. Mnichov. SIEMENS, 2017.

[9] LERMAN, Julian. Programming Entity Framework. 2nd ed. United States of America: O'Reilly Media, 2010. ISBN 978-0-596-80726-9.

[10] SELLS, Chris a Ian GRIFFITHS. Programming WPF. 2nd ed. United States of America: O'Reilly Media, 2007. ISBN 0-596-51037-3.

[11] KREIBICH, Jay A. Using SQLite. United States of America: O'Reilly Media, 2010. ISBN 978-0-596-52118-9.

Seznam symbolů, veličin a zkratek

LAD	Ladder diagram
ST	strukturovaný text – Structured Text
UDT	Uživatelsky definovaný datový typ – User Defined data Type
MVVM	Model-View-ViewModel
OB	Organizační blok – Organization Block
FB	Funkční blok – Function Block
FC	Funkce – Function
DB	Datový blok – Data Block
TIA	Totally Integrated Automation Portal
API	Application Programming Interface
PLC	Programovatelný logický automat – Programmable Logic Controller
IDE	Integrated Development Environment
XML	Extensible Markup Language
HMI	Human Machine Interface
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language

Seznam příloh

A Vytvořený projekt

- `/` — Kořenový adresář přiloženého projektu
  - `Test1` — Adresář vytvořeného projektu aplikace
    - `DATA` — Adresář dat využívaných aplikací při práci s projektem
      - `Blocks` — Adresář využívaných bloků
        - `1fea2e6f-fe0f-4273-9bce-51b835acf47a` — Adresář využívaného bloku
          - `1fea2e6f-fe0f-4273-9bce-51b835acf47a.xml`
      - `TagTables`
      - `UDTs`
    - `TEST_1.db` — Projekt vytvořený a otevíratelný z aplikace
  - `Test1Projects` — TIA portal projekty vytvořené aplikací
    - `TEST_1` — TIA portal projekt vytvořený aplikací
      - `...` — Struktura projektu TIA portal

B Obsah přiloženého CD

Přiložené CD obsahuje ve složce Práce vytvořenou práci v programu LATEX. Dále vytvořenou aplikaci ve složce AVIC.Openness a vytvořený testovací projekt ve složce Priloha_projekt